

Building Business Applications with Microsoft Silverlight

Microsoft Corporation – April 2010

***Abstract***: Silverlight has a wealth of features directly applicable for developers building business applications. At the same time, the enhanced tooling provided by Microsoft Visual Studio® and Microsoft Expression Blend® takes much of the pain out of building business applications due to the quality of the design-time experience. This paper describes how to leverage Silverlight (and the associated tooling) to build business applications. The paper presents an overview of the rationale, architectural approaches, patterns, and toolkits to consider when building business applications with Silverlight.

Contents

[Introduction 3](#_Toc258228172)

[What Is a Business Application? 3](#_Toc258228173)

[Why Consider Silverlight for Building Business Applications? 4](#_Toc258228174)

[Architectural Considerations 5](#_Toc258228175)

[HTML / ASP.NET / PHP Application Architecture 5](#_Toc258228176)

[Rich Internet Application Architecture 6](#_Toc258228177)

[WCF RIA Services 6](#_Toc258228178)

[Displaying Data on the Client 8](#_Toc258228179)

[Handling Large Amounts of Data 9](#_Toc258228180)

[Capturing Data on the Client 9](#_Toc258228181)

[Validating User Input 10](#_Toc258228182)

[Validation Attributes 10](#_Toc258228183)

[Shared Custom Validation Logic 10](#_Toc258228184)

[Microsoft Extensibility Framework (MEF) 11](#_Toc258228185)

[Security Considerations 12](#_Toc258228186)

[Role Checking 12](#_Toc258228187)

[Accessing Local Resources 14](#_Toc258228188)

[Printing 15](#_Toc258228189)

[Patterns 16](#_Toc258228190)

[Model-View-ViewModel 16](#_Toc258228191)

[Dependency Injection 17](#_Toc258228192)

[PRISM (Composite Application Guidance for WPF and Silverlight) 18](#_Toc258228193)

[Conclusion 20](#_Toc258228194)

[Additional Resources 20](#_Toc258228195)

[Silverlight Resources 20](#_Toc258228196)

[RIA Services Resources 20](#_Toc258228197)

[PRISM Resources 20](#_Toc258228198)

[MEF Resources 20](#_Toc258228199)

[Printing Resources 20](#_Toc258228200)

[Legal Notice 21](#_Toc258228201)

# Introduction

Silverlight has rapidly evolved since its original release in 2007. The initial emphasis for Silverlight was on providing rich media experiences, enabling designers and developers to add rich degrees of interactivity, media and animation to their Web sites. Silverlight 3 extended the capabilities of the platform, adding many new features essential for business application development. The rich tool support provided by Visual Studio and Expression Blend also helps you to speed up your business application development and lower the cost of development. Silverlight 4, the current version, adds even more features making Silverlight a compelling choice of technology for building business applications.

## What Is a Business Application?

So what do we mean by the term “business application”? A business application has a distinct purpose or goal, and generally helps a company to save money or time, or perhaps helps them to improve customer satisfaction. Common examples might include CRM applications, time-tracking and expense claim applications, e-commerce and e-tail experiences, purchase request applications, and HR employee management systems. This style of application has a number of common characteristics.

Business applications:

* **Are data-driven**. Data is all-important, and the application needs to be able to query, shape, and display data in different ways. The application also needs to be able to take input from the user via form fields, validate it, and then save and update it appropriately.
* **Require robust data validation**. To ensure the best user experience, validation of input data needs to occur on the client with immediate feedback for the user. For added security, the validation must also be provided on the server.
* **Need to be secure**. The data displayed by the application typically needs to vary depending on who is accessing the application. Authentication (in other words, knowing who is accessing the application) is essential. Also, the functionality offered by the application often needs to vary depending on the user’s role.
* **Must present a professional, efficient, and enjoyable user experience**. A professional interface that matches corporate branding requirements and enables users to get their jobs done efficiently and effectively is generally more important than fancy visual effects. A business application also needs to visualize data in many different ways using a variety of different control types, ranging from simple text boxes, combo boxes, and data grids to more sophisticated controls providing charts and reports.
* **Often need access to local resources**. Many business applications also need access to local resources including the local file system, other client-side business productivity applications such as Microsoft Office Excel® or Microsoft Office Outlook®, printers, microphones, webcams, the clipboard, and so on.
* **Depend on integration with external and internal services**. Many business applications need to access existing services and applications to provide critical data essential for the operation of the business application. These may be accessed locally over the internal network or remotely over the Internet. Access is typically via REST or SOAP calls.

## Why Consider Silverlight for Building Business Applications?

Users of business applications expect rich, functional, and well-performing interfaces such as those typically provided by the desktop applications they are used to. With the latest business application-focused enhancements to Silverlight, the rich capabilities of its user interface, and the associated tooling provided by Visual Studio and Expression Studio, you can now rapidly build this type of application and enjoy the benefits of Web delivery and deployment.

The following features in particular make Silverlight a viable and compelling technology for building business applications:

* **WCF RIA Services**. WCF Rich Internet Application (RIA) Services and Visual Studio 2010 provide an elegant solution for handling the transmission of data across the tiers of your application, data validation, and change tracking. In doing so, they provide a unified model for client-side and server-side development, making a traditionally difficult job for the developer much easier.
* **Rich data controls**. Silverlight provides a rich library of over sixty controls complimented by open source and vendor control packs. The new, functionality-rich data bound controls such as the **DataGrid**, **ContentControl**, **DatePicker**, and charting controls provided by the Silverlight Toolkit make it much easier to display data in an attractive manner. New controls such as the **RichTextArea** control make it much easier to capture formatted text input. Working with large quantities of data and handling data paging is also much easier with the **DataPager** control, which largely automates this job.
* **Page navigation**. Silverlight provides a page navigation framework, making it much easier to support navigation between pages. You can control how and when page navigation should interact with the browser history journal, and it also provides URI mapping, enabling you to use meaningful URIs to navigate between the different pages in your application.
* **Out-of-browser support**. The out-of-browser feature enables your users to continue working while disconnected from the network. For example, you could write an application that synchronizes their changes back to the server the next time they connect. Out-of-browser applications can also run as trusted applications with more privileges, making tasks requiring local resource access possible. For example, you can access the local file system, interact with Office applications, or have full keyboard access in full-screen mode.
* **Shared code validation**. The validation feature support means you can write your validation logic once, and have the code automatically reused in the client and on the server. You can also declaratively apply basic validation support, such as length and range checks.
* **Local resource access**. Silverlight enables out-of-browser Silverlight applications to access local resources, such as the file system and clipboard, as well as other local applications and devices that provide a COM interface. Using the COM support enables your Silverlight business applications to integrate with Office applications such as Excel or Outlook and perform many other tasks previously not possible from Web applications.
* **Printing support**. Silverlight printing support enables you to print whole pages and specific data items directly from your business applications.
* **Command support**. Silverlight now provides support for commands in the same way that WPF does for desktop applications. Commands provide a way to data-bind specific operations in your application to controls in your application’s user interface. This command approach supports patterns like Model-View-ViewModel (MVVM) that promote code separation, maintainability, and testability.
* **Drag and drop**. Silverlight applications can now expose drop targets, enabling users to drag files to your Silverlight applications. For example, a user could drag and drop an Excel document onto a Silverlight application and have the application automatically display its data in a DataGrid control.
* **Right-click support**. You can now create your own right-click pop-up menus to further enhance the user experience and provide easy access to context-specific features and functions.

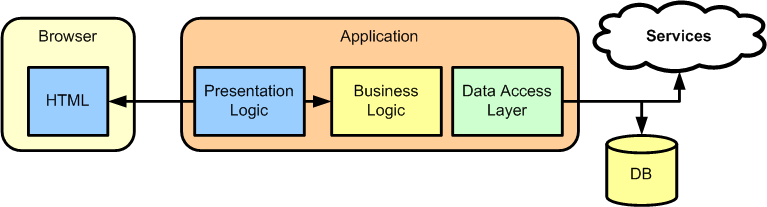
So, while many features essential for business applications are now available, how should you structure your Silverlight application to best take advantage of them? How should you perform data access? How should you pass data over the network between your rich UI and database? Establishing and understanding the right architecture up front is essential.

# Architectural Considerations

Silverlight application architecture shares much in common with traditional HTML/ASP.NET/PHP applications, with one important difference. When you use Silverlight, the presentation logic is located solely on the client in your user’s browser.

## HTML / ASP.NET / PHP Application Architecture

Consider the architecture of a typical HTML-based Web-based application such as one you might create using ASP.NET. This is shown in Figure 1.

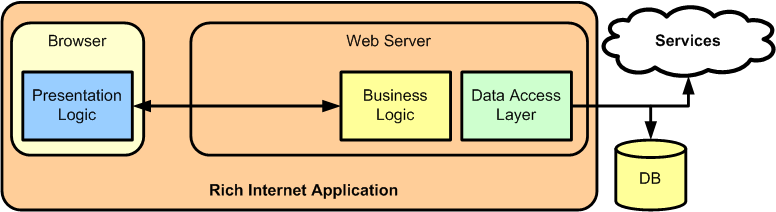


***Figure 1: HTML / ASP.NET / PHP Application Architecture***

With this very common model, the application is divided into three tiers: presentation, business and data. In the conventional ASP.NET/PHP model, the presentation logic on the server is responsible for generating the necessary HTML markup, which is sent to the client’s browser to render the application’s user interface. In many systems, the presentation and business logic, and often the data access logic, is more closely coupled than Figure 1 would suggest, but a clear separation into distinct tiers is desirable and offers many benefits including flexibility, maintainability, and scalability. With AJAX technology, the richness of the user interfaces presented by this type of application has significantly increased, but Silverlight enables you to take the richness and functionality to another level.

## Rich Internet Application Architecture

The Rich Internet Application (RIA) model offered by Silverlight changes things a little by providing a hybrid model that combines the power of Silverlight for displaying a very rich user experience (more akin to a traditional desktop application) on the client, with the conventional three-tier Web model. Figure 2 shows the RIA architecture.



***Figure 2: Silverlight / RIA Application Architecture***

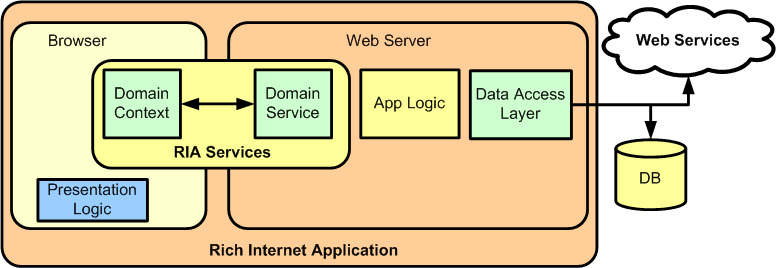
Notice how the presentation tier is now completely client-side (running in the Silverlight plug-in inside the user’s browser). In this model, the client is really an extension of the server and you should think of the RIA as a single logical application. This raises a key question: How should you handle the communication between client and server, across the trust boundary that is the network?

## WCF RIA Services

The introduction of WCF RIA Services (hereafter called RIA Services) provides a solution to this issue, and in doing so abstracts much of the underlying complexity of communicating across a trust boundary from client to server. The fundamental objective for RIA Services is to provide a coherent, end-to-end view of an application, connecting the presentation views on one side with the database on the other. This, together with the fact that RIA service abstracts away the detail of having to deal with asynchronous communication (something Silverlight relies upon), greatly simplifies the overall development task.

Central to the RIA Services approach is a server-side *domain* *service* class. This exposes a set of methods callable over the network (called operations) required by the presentation logic, such as operations for querying and returning sets of data, for updating existing data and inserting new data into the backend data store through the data access layer.

A domain service on the server is exposed to the client-side presentation tier through the *domain* *context* class (automatically generated from the domain service class by the Visual Studio tools when you build your project). The RIA Services solution layered on top of the RIA application architecture is shown in Figure 3.



***Figure 3: WCF RIA Services, the domain service and domain context classes***

The domain context class is responsible for providing access to the data via a series of public operations (methods). While WCF RIA Services does not mandate which data access technology you must use, a convenient approach is the ADO.NET Entity Framework, which enables you to program against the data model and which takes care of the object to relational mapping for you.

**NOTE**: RIA Services works with any data access approach. For example, you might choose to use NHibernate or a custom data access architecture. Regardless of the data access approach, you can still expose your data by using RIA services.

The following code snippet shows what a typical domain service class looks like. This code is automatically generated by Visual Studio when you add a new item of type Domain Service to your project.

[EnableClientAccess()]

public class ProductDomainService : LinqToEntitiesDomainService<NorthwindEntities>

{

public IQueryable<Product> GetProducts()

{

return this.ObjectContext.Products;

}

public void InsertProduct(Product product)

{

if ((product.EntityState != EntityState.Added))

{

if ((product.EntityState != EntityState.Detached))

{

this.ObjectContext.ObjectStateManager.ChangeObjectState(product,  
 EntityState.Added);

}

else

{

this.ObjectContext.AddToProducts(product);

}

}

}

. . .

Notice how this example is using the Entity Framework for data access and as a result the domain service class derives from the **LinqToEntitiesDomainService** base class. The class is also annotated with the **EnableClientAccess** attribute. This defines that the domain service class should be projected to the client and be made available via the associated domain context class. This client-side class is automatically generated for you when you build your project inside Visual Studio.

In the above example, the domain service class has been generated from a **Product** entity, generated from the Microsoft SQL Server® Northwind database. By default, any newly created domain class exposes a set of CRUD operations, including:

* A public Get method (**GetProducts** in this example) that returns an **IQueryable** interface, in this case encapsulating a set of **Product** objects. You can easily add additional LINQ statements to filter, sort, or restrict the returned product set in some way depending on your business logic requirements.
* Insert, Update, and Delete methods for performing other CRUD style operations.

You can also add your own custom methods to the domain service class. These can include custom business logic and custom validation. For example, you may need to provide a **GetProductByID** method that accepts a product ID as an input parameter and returns specific product details.

# Displaying Data on the Client

The server-side domain service is presented to your Silverlight project as a domain context object. Visual Studio automatically generates the code for the domain context class for you based on your domain service class. The domain context class exposes a similar set of CRUD operations for the Silverlight client to call, enabling data to be queried and retrieved, inserted, updated, and deleted against your data source. This capability of replicating the server-side operations to the client-side Silverlight code helps to streamline and simplify the development process.

To display the data on your Silverlight UI, you can choose from a variety of controls including **TextBox**, **ComboBox**, **ListBox**, or **DataGrid**, all of which support data binding directly to the entities collection on the data context. For example, the following code snippet shows how to use the domain context to retrieve a set of data and bind it to the **DataGrid** control.

// Create a new domain context

var context = new ProductsDomainContext();

// Data bind the Products collection exposed by the domain context

// to the data grid control

productDataGrid.ItemsSource = context.Products;

// Load the data by calling the GetProductsQuery method

context.Load(context.GetProductsQuery());

When **GetProductsQuery** is called on the client-side domain context, Microsoft Windows® Communication Foundation (WCF) service communication is used to call the equivalent domain service operation on the Web server, retrieve the data, and load it into the client-side context, where it is helpfully exposed via an Entity collection property.

In addition to being able to programmatically establish the connections between your data and the UI controls, you can also use the designer support in Visual Studio by dragging controls directly from the Data Sources window onto your design surface. This establishes all of the data bindings for you without you needing to write any explicit code.

### Handling Large Amounts of Data

If you need to handle large data sets, such as products from a product catalog, orders from a purchase order system, or employee records from an HR system, you need to consider data paging, which is presenting a single page of data at a time while allowing the user to perform sorting and filtering across the entire data set.

Silverlight provides the **DataPager** control to greatly simplify this task. To add data paging to your application, you can simply drag and drop the **DataPager** control onto your design surface, typically beneath a **DataGrid.** You then bind the **DataPager** to the data source exposed by the domain context and subsequent data retrieval is handled a page at a time.

Even with data paging, network latency and data processing time can cause delays. To provide a visual cue to your users that activity is occurring, you can use the **BusyIndicator** control to provide feedback. This control, along with many other data visualization controls, is available in the Silverlight Toolkit which you can download from <http://silverlight.codeplex.com>.

# Capturing Data on the Client

Silverlight supports controls to handle all of the common forms-based data input tasks your business application might need. Some rich controls, including **Calendar** and **DatePicker**, enable you to quickly construct data input forms. Using the new controls and also the new tooling provided by Visual Studio 2010 can greatly improve your development productivity and also enhance your application’s user experience.

As an example of the productivity enhancements provided by Visual Studio 2010, you can now use options from within the data sources window to auto-generate large chunks of your user interface based on the data exposed by your domain context classes. The data sources window lists the domain context classes in your application and lets you specify the type of control you’d like to use to display each data property, such as a check box, radio button, edit box, combo box, image control, and so on. Alternatively you can accept the defaults that Visual Studio applies. You can also decide whether or not you want to display the data as a master list or a details view. This enables you to display tabulated lists of data together with forms that enable you to edit and update that data.

The enhanced tooling provided by Visual Studio 2010 and Expression Blend takes much of the pain out of building business applications due to the quality of the design-time experience.

# Validating User Input

Business applications need to validate their user input in a variety of ways, ranging from simple type, length, and range checks to more sophisticated, custom validation that depends on business rules. In a Silverlight application (or indeed any Web application), the validation needs to occur in two places:

* **On the client**. To provide a rich user experience and instantly present well-formatted validation error messages to the user next to the input field containing the invalid entry.
* **On** **the** **server**. For added security, in case a malicious user bypasses your Silverlight client application and attempts to call your backend services directly.

### Validation Attributes

RIA Services enables you to leverage types from the **System.ComponentModel.DataAnnotations** namespace that you can apply directly to properties on your entity data model classes to perform common validation checks. You add them in your server project and they are automatically propagated at build time, to the associated client-side classes. **DataAnnotations** that provide property validation include **DataTypeAttribute**, **RangeAttribute**, **RegularExpressionAttribute**, **RequiredAttribute**, and **StringLengthAttribute**.

NOTE: You can use **DataAnnotations** with or without RIA Services.

An example is shown in the following code snippet.

[Required(ErrorMessage=”You must provide a surname”)]

[StringLength(50, ErrorMessage=”Surname cannot exceed 50 characters”)]

public string Surname

{

get { return Surname; }

set { … }

}

### Shared Custom Validation Logic

For more sophisticated and custom validation logic, you can use the **CustomValidationAttribute**. You can apply custom validation to a single property or to an entire entity class, enabling you to provide entity-level validation. In order to write the custom validation code only once, yet enable the validation logic to be applied on both the client and server sides, you add your validation class code to the Web server project in a file with the .shared.cs or .shared.vb file extension. At build time, the shared code is automatically propagated to the Silverlight project in order to provide the client-side validation.

For more information about Silverlight data validation, see “Data Validation with Silverlight 3 and the DataForm” at <http://msdn.microsoft.com/en-us/magazine/ee335695.aspx>.

## Microsoft Extensibility Framework (MEF)

Business applications, like most applications, need to evolve over time as businesses change and evolve and new requirements for new functionality emerge. You need an effective way to enable an application to be incrementally extended over time, without having to change the application’s existing codebase. This is really important, because by avoiding any changes to the existing code, you do not have to worry about introducing new bugs and having to go through a lengthy and costly retesting cycle.

The Managed Extensibility Framework (MEF), a new library in Silverlight 4, can help you to simplify the design of extensible applications. MEF provides a way for you to assemble applications from extensions and be able to dynamically add extensions to your application over time. This ability to plug new functionality into your applications without having to touch the existing application’s code provides significant benefits for application maintainability. It also enables third parties to extend your application after it is deployed, adding new features and functionality. MEF also enables you to separate concerns in your application, a benefit over and above extensibility.

For more information about MEF, see “Building Composable Apps in .NET 4 with the Managed Extensibility Framework” at <http://msdn.microsoft.com/en-us/magazine/ee291628.aspx>.

# Security Considerations

Business applications need to be protected, as they often deal with sensitive and restricted data and functionality that should only be displayed to specific individuals or specific types of user, such as managers. Also, for legislation compliance, you usually need to know who is accessing your data. When building your Silverlight business applications, you have access to the ASP.NET membership and role management system to provide authentication and authorization. Note that the ASP.NET Profile system is also available to help you provide personalization features for your users – for example, to track user specific preferences such as color choices or other application-specific settings.

RIA Services supports both Windows Integration authentication and Forms authentication. By using Windows authentication, you can authenticate users with their regular Windows Active Directory domain accounts providing a single sign-on experience for them. With Forms authentication, you authenticate users with custom usernames and passwords maintained in a separate user store, most often a SQL Server database.

To restrict access to your domain service and require authenticated access, you can add the **RequiresAuthentication** attribute to your domain service class.

[EnableClientAccess()]

[RequiresAuthentication]

public class ProductDomainService : LinqToEntitiesDomainService<NorthwindEntities>

When a user now attempts to access your Silverlight application, an error will result when the application attempts to call the domain service. In response to the error, you would typically display the login dialog box. Note that the Silverlight 4 Silverlight Business Application template automatically provides login and register dialog boxes together with a login link from the application’s home page.

### Role Checking

Once authenticated, you often need to check the role membership of the authenticated user and perform different operations depending on the type of user accessing your application. For example, you might have some sensitive data that should only be displayed to managers.

You should restrict access to this type of sensitive data at source on the server, and not send it over the network to unauthorized users. The following code snippet shows this approach.

[RequiresAuthentication]

[EnableClientAccess()]

public class EmployeesDomainService:LinqToEntitiesDomainService<NorthwindEntities>

{

public IQueryable<Employee> GetEmployees()

{

foreach (var e in this.ObjectContext.Employees)

{

if (!this.ServiceContext.User.IsInRole("Managers"))

{

e.Salary = null;

}

else if (e.Title.Contains("President"))

{

e.Salary = null;

}

}

return this.ObjectContext.Employees;

}

Notice how the standard data query operation on the domain context (**GetEmployees** in this example) has been extended to include some additional role-checking logic that determines whether or not an employee’s salary should be returned to the client.

NOTE: While RIA services makes your development easier, Silverlight provides some of its own credential-based security features. These and the ASP.NET membership system can be used with or without RIA services.

# Accessing Local Resources

Many business applications require access to local resources, such as the user’s local file system, other applications running on the user’s computer, or other devices such as webcams and microphones. Traditionally, there has only been limited support for local resource access from Web applications, due to the restricted sandbox with limited privileges in which Web applications run.

Silverlight 3 introduced out-of-browser applications, which a user can choose to install locally and run directly from the Start menu or desktop. Appropriately designed out-of-browser applications can also run offline, without requiring a connection to the Web server. Your application can use isolated storage to store offline data, use network detection to check for a network connection, and then re-synchronize data back to the server when a connection is available.

For security reasons, out-of-browser applications run as partial-trust applications with limited privileges. Silverlight 4 extends the capabilities of out-of-browser applications by enabling them to run as a trusted application with additional privileges. When installing a Silverlight 4 out-of-browser application locally, your user is prompted about the potential risks associated with allowing this and users should be educated to only install trusted applications from known sources.

Regular browser-based Silverlight applications can:

* Access the user’s webcam. A security registration application at the reception desk of an office building could use the webcam to take a snapshot picture of a visitor to save or print.
* Access the user’s microphone. A user could add voice annotations to a set of data presented by an application to provide feedback.
* Access network resources and services in separate network domains. This is important if your application needs to access data from other servers and services located elsewhere in your organization.

When running as an out-of browser application, your Silverlight applications can:

* Load HTML content from a Web page by using the WebBrowser control.
* Display pop-up notifications near the Windows system tray.

When running as a trusted out-of-browser application, your Silverlight applications can:

* Access files on the local file system, for example, to read and write local data.
* Access Office applications via COM interop. For example your application could provide an export to Excel option.
* Access any other devices or applications that provide a COM interface. Many Windows functions and capabilities are exposed by COM interfaces. Being able to make COM calls enables you to extend your applications to leverage the full power of the Windows operating system.
* Use the full keyboard while running in full-screen mode. This is important for certain types of interface where the full screen view is needed.

# Printing

Printing is a common requirement for many line-of-business applications. Silverlight now has printing support that enables you to present print previews and to print selected parts or all of your application’s interface or data displayed. Printing is controlled by the **PrintDocument** class, which exposes a number of events that are called to ask you about how to print individual pages. Printing a document is as simple as setting the document name (this is what shows up in the print spooler), handling events such as **StartPrint** and **PrintPage** events, and then starting the printing process by calling the **Print** method of the **PrintDocument** object.

The following code snippet shows the basic approach.

private void btnPrint\_Click(object sender, RoutedEventArgs e)

{

PrintDocument pdoc = new PrintDocument();

pdoc.DocumentName = “Employee Listing”;

pdoc.PrintPage += (s, args) =>

{

// Assign the XAML element to be printed. This can be any UIElement-derived

// element that you need to print.

args.PageVisual = this.employeeDataGrid;

// Controls whether or not to call again for another page

e.HasMorePages = false;

};

pdoc.Print();

}

Typically, you would print an entire page or a specific control such as a **DataGrid** or a container for other elements if you need to print part of a page. You need to provide a print button in your application, to which you attach your printing code. In response to the user clicking the application’s print button, the **PrintDocument** class accesses and displays the operating system specific **Print** dialog box, enabling your user to choose a printer, specify print preferences, and so on before sending the data to the printer.

For more information about Silverlight printing, see <http://silverlight.net/learn/videos/all/printing-api-basics/>.

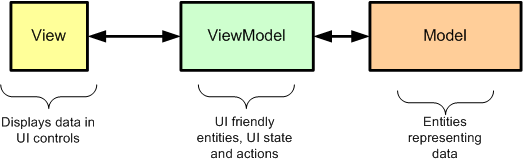
# Patterns

There are a number of design patterns that you should consider when building your line-of-business Silverlight applications, in order to better structure and modularize your applications and make them inherently more testable, extensible, scalable, and maintainable. While they are not mandatory for building Silverlight applications, patterns such as Model-View-ViewModel (MVVM) and Dependency Injection are highly recommended.

## Model-View-ViewModel

The MVVM pattern is fundamentally about the separation of concerns within your code base. For example, MVVM enables you to separate your Views (the presentation of the application’s interface to the user) from the data and logic. Designers can design the View in Expression Blend while the ViewModel can be developed in Visual Studio. One of the core benefits of MVVM is that it improves the testability of your code by moving application logic out of the View and into the ViewModel. The View is typically difficult to test, because you need your test harness to drive the mouse and keyboard, or you need to fall back to manual, labor-intensive tests. The ViewModel is easier to test because you can drive it from test code. So the more you move from the View to the ViewModel, the easier the testing gets. This also helps when it comes to debugging your code.

Your Views, view models, and Models are all separate entitles that are loosely coupled. They communicate with one another, but only through defined interfaces.



***Figure 4: Model-View-ViewModel Pattern***

So how is the View different from the ViewModel and the Model?

* **View**. The View contains the controls and behaviors that make up the user interface. It can also contain animations, navigation aspects, themes and other interactive features used to render the visual interface. The whole View should be defined declaratively (in XAML) where possible in order to benefit from (for example) the Expression Blend and Visual Studio designer support. The View also contains the declarative bindings (again in the XAML) that identify the data items that will be presented to the user. The bindings point to the names of the data item properties, but do not have awareness of where those properties are nor where they come from. You should keep the code in the View and its code behind file to a minimum.
* **ViewModel**. The ViewModel represents the data for the View, and handles the communication between View and Model through bindings. It contains no user interface elements, but drives the Model, and provides the entry points into the hierarchies of data. It can also shape and combine the data in different ways so that the View can bind to it. ViewModel classes typically support the **INotifyPropertyChanged** and **INotifyCollectionChanged** interfaces as ways to notify the View when some aspect of the data has changed. The ViewModel also listens to changes from the View through its bindings.
* **Model**. The Model is simply the data, ignorant of everything else within your application. For example, in an HR application, it would contain your Employee entities. The Model does not need to know where the data comes from. It could come from a WCF service, WCF RIA Services, a RESTful service (such as Twitter, an RSS Feed, or Amazon), a SQL Server database and so on. The Model may also contain validation.

## Dependency Injection

Dependency injection is another pattern central to building testable, modular, and decoupled applications and it is used within the MVVM approach to provide loosely coupled connections between the Model, View, and ViewModel. Without dependency injection, these connections would need to be hard-coded. Dependency injection lets you insert or “inject” the classes to use. For example, you can use different classes when unit testing as opposed to at run time.

The key to dependency injection is to program against an interface rather than a concrete type or class. This enables you to switch in and out different implementations, such as mock implementations for testing purposes. Consider an interface used to expose a model for a time-sheet recording application, as in the following code.

public interface ITimeSheetSystem  
{  
 ObservableCollection<Employee> GetEmployees();  
}

By programming against the interface **ITimeSheetSystem**, different implementations can be provided including a mock class used when running unit tests.

public class MockTimeSheetSystem : ITimeSheetSystem  
{  
 public ObservableCollection<Employee> GetEmployees()  
 {  
 return new ObservableCollection<Employee> { new Employee{ID=1,Name=”Sam Smith”,   
 Age=25, Department=”Sales”}};  
 }  
}

## PRISM (Composite Application Guidance for WPF and Silverlight)

PRISM is a toolset that you can use to build applications that are testable, modular, and extensible. It enables you to build composite user interfaces for both WPF and Silverlight applications, and enables you to support application development across multiple teams. PRISM separates the UI-specific elements from the presentation and business logic elements of the application, and shows how you can test them independently. It also shows how you can use the MVVM pattern to share non-UI code and components between Silverlight and WPF applications. PRISM also includes other patterns to support the development of modular, extensible applications.

PRISM provides the following components:

* **Shell**. This contains all of the views that will subsequently be loaded. It is a container for the different UI regions you need to display. You can think of this as your main screen for the user where all other controls and elements are presented.
* **Regions**. The shell provides regions, or named placeholders, into which you can place views. Your user interface may consist of one or many regions. Think of regions as the places where elements that the user will interact with are presented.
* **Modules**. Modules enable you to build your application as a discrete set of modular blocks independent of one another. They provide a way for you to partition your solution. Each module is an independent entity that does not directly reference the other modules. If needed, you can subsequently use commands or events to communicate between them.

Modules are really important for business application development because there are often phases of a project where different individuals or teams are working on parts of the overall application. These individuals or teams can work on the modules without knowing very much about what the others are doing. PRISM helps integrate these modules even if they are released at different times. It also enables your application to load modules on demand.

* **Dependency Injection Container**. PRISM uses the Unity Dependency Injection container. The container is the object that coordinates the creation of objects and the objects that they depend on. Dependency injection is a critical ingredient for building testable, modular and decoupled frameworks and applications. For more information about Unity, see <http://msdn.microsoft.com/en-us/library/dd203101.aspx>.
* **Bootstrapper**. The bootstrapper starts your application and loads the shell (the main UI container). It also registers and loads any necessary modules and provides a way for you to run other initialization tasks you may need to perform.
* **Event** **aggregation**. Provides a loosely-coupled way to publish and subscribe to events. This enables different components within the application to communicate with each other without needing a direct reference.

For more complex Silverlight line-of-business applications, providing a lot of functionality and facing the challenge of presenting many different views of data to the user, using PRISM offers some significant benefits by imposing discipline and structure to your software and by providing a set of very useful tools and techniques. PRISM provides a number of distinct items, but you can choose to use each of them selectively. You can use just the pieces of PRISM you want (such as commanding) without using all of them.

For more information about PRISM, see <http://msdn.microsoft.com/en-us/library/dd458809.aspx>.

# Conclusion

Business applications have a set of functional requirements and capabilities that have traditionally demanded a rich client and desktop-application-based solution. However, this brings with it the deployment and upgrade challenges associated with desktop software. Web software delivered through a browser has none of these deployment issues, but up to now has been unable to provide a rich enough user experience demanded by business application users. Neither has it been able to utilize fully the functionality and features of the host operating system, including local resource access that is required by most business applications.

Silverlight together with its frameworks and tool support now provides a viable alternative and one that brings together the best of the desktop and Web-based models of application development. The enhanced tooling provided by Visual Studio 2010 and Expression Blend 3 takes much of the pain out of building business applications, due to the quality of the design-time experience. This helps you to speed up your development, and as a result helps you to build business applications at lower cost. Silverlight is more productive as a platform for CRUD applications than any other because you no longer have to spend time on tedious work to marshal data between business objects and controls.

# Additional Resources

For more information about using Silverlight to build line-of-business applications, see:

### Silverlight

* www.microsoft.com/silverlight/developer
* <http://www.silverlight.net>
* “Data Validation with Silverlight 3 and the DataForm” at <http://msdn.microsoft.com/en-us/magazine/ee335695.aspx>
* Silverlight TV <http://channel9.msdn.com/shows/SilverlightTV/>

### Visual Studio

* [www.microsoft.com/visualstudio](http://www.microsoft.com/visualstudio)

### Expression Blend and Sketchflow

* [www.microsoft.com/expression](http://www.microsoft.com/expression)

### RIA Services

* <http://silverlight.net/getstarted/riaservices/>

### PRISM

* <http://msdn.microsoft.com/en-us/library/dd458809.aspx>
* <http://www.codeplex.com/CompositeWPF>

### MEF

* <http://www.codeplex.com/mef>
* Building Composable Apps in .NET 4 with the Managed Extensibility Framework” at <http://msdn.microsoft.com/en-us/magazine/ee291628.aspx>

### Printing

* <http://silverlight.net/learn/videos/all/printing-api-basics/>

# Legal Notice

This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2010 Microsoft Corporation. All rights reserved.

Microsoft, Expression, Silverlight, the Silverlight logo, Windows Server, the Windows logo, Windows Media, and Visual Studio are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.