



**MEAP Edition  
Manning Early Access Program**

Copyright 2008 Manning Publications

For more information on this and other Manning titles go to  
[www.manning.com](http://www.manning.com)

Please post comments or corrections to the Author Online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=443>

## **Hello! Silverlight Table of Contents**

- 1. A Silverlight in the Storm**
- 2. XAML Basics**
- 3. Animations and Data**
- 4. Silverlight Controls**
- 5. Talking to the Outside World**
- 6. Data Services**
- 7. Writing Games in Silverlight**

# 1

## *A Silverlight in the Storm*

Silverlight is a case of the right technology at the right time. This is a time of convergence. Companies want to be able to leverage the experience of their developers when developing Rich Internet Applications (RIAs). Developers want to write applications using the development tools and languages that they are comfortable with. Desktop application developers, who previously may have felt that web development was out of reach, can feel at home developing Silverlight applications. Silverlight is a shining beacon in a storm of web platforms from a myriad of vendors using a dizzying array of technologies.

If you're like me, and have years of experience writing Windows applications, then you would probably agree that writing a web application is kind of like writing in a dialect that is not completely familiar. The stateless model of web applications and the session management model is something I've always struggled with, and if you want to do anything interesting with the user interface, it usually requires you to write a bunch of client script code and many more hours of development than it would take to do the same thing in a client application.

Of course, Windows application development has many of its own issues. Deployment and updating of the application is difficult (although this is a lot easier using ClickOnce). For most modern Windows applications, the prerequisites that need to be installed on the machine can be prohibitive. If you write a mainstream application using the .Net 2.0 Framework, and it's not already installed on the target machine, it can mean a 20+ Megabyte download and a half hour to install. This isn't the ideal first impression for your users. Then let's say you want to provide some of the same functionality in a browser. You'll end up having to write a web version anyway, and it will most likely be a different code base because of the differences in the two platforms.

Windows application development received a big boost with the release of Windows Presentation Foundation (WPF) which provided a powerful vector graphics library, an XML-based markup language called XAML to lay out user interfaces, and a totally revamped

multimedia library. Windows Vista supports WPF out of the box, but Windows XP users need to install the .Net 3.0 Framework to run WPF applications. As the Windows Vista installed base increases, WPF applications will become more and more common.

So where does Silverlight fit into all of this? Silverlight combines some of the best features of WPF with the ease of deployment and cross platform reach of web based applications. Silverlight 2 contains the Silverlight Common Language Runtime (CLR) which is a smaller version of the .Net Framework 3.0. The Silverlight CLR runs on the PC and Mac, is only 4 Megabytes to download, and installs in seconds. This means that with Silverlight 2, developers can use any .Net based language, including C# and VB.Net, to develop their web applications.

Silverlight is not alone as an RIA technology, and is actually one of the new kids on the block. Let's take a look at some of the benefits that RIAs bring to the table and some of the similarities and differences between how these RIA technologies work.

## ***1.1 The rise of the RIAs***

RIA was originally coined by Macromedia in 2002 in reference to applications built using their Flash technology. Although Adobe invented the terminology, other vendors had been working with their own technologies to achieve similar goals. For example, in the late 1990s, Microsoft released a technology called Remote Scripting which shared some characteristics with modern RIAs, and could also be considered a precursor to Ajax. It allowed for server code to modify the web page's contents without reloading the page.

Applications that are categorized as RIAs typically run inside the browser while delivering an experience similar to traditional desktop application. The code to handle the user interface runs on the client and data is retrieved as needed from the web server. Because RIAs are browser based, they normally do not have full access to local resources, but can store local data in a sandbox on the client. The following sections discuss the features that RIAs have to offer, and how those features can help to deliver a new level of user experience.

### ***1.1.1 Advantages of RIAs over traditional web development***

RIAs were born out of the desire to increase responsiveness of web applications and to provide a user interface that is either difficult to impossible to provide using traditional web technologies such as HTML and CSS. Since a page refresh isn't required each time the data changes, only the data needs to pass between the server and the client, instead of the entire HTML layout to render the page. This can significantly reduce the amount of information that has to travel over the wire.

Another area where these applications can offer an advantage is with presenting multimedia. RIAs typically provide seamless integration of video, images, and sound, where in the past, videos would typically run in a separate player window or in a square box on the page. Additionally, this would require the software to already be installed on the machine to play the video, most commonly Windows Media Player or Quicktime. In many cases the server would have two or more copies of the video, one for each video player type, and the user would either have to choose which video format to view, or the site would have to determine which to serve up based on which player the client supported.

I'm always impressed with what some sites are able to do using traditional web technologies, and the creativity involved in thinking outside the box. I believe that software development can be an art form, and web development has its share of artists.

### **THE REMBRANDTS OF THE WEB**

A few months ago, I attended an art auction. I don't know that much about art, but I enjoy it and I'm always interested in learning more. On this occasion, the art auction featured some pieces from Rembrandt van Rijn, a famous artist and the most important artist in Dutch history. The auctioneer gave us a bit of a history lesson, and explained how Rembrandt painted to pay the bills, but his passion was for etchings. He would create his etchings by starting with a metal plate covered with wax, and would methodically remove the wax to create the image that he desired. Rembrandt was able to take this medium that was previously mainly used for printing, and did things with it that were never conceived of before, and elevated the medium to a new level. Never before had someone created etchings with the level of detail and precision that Rembrandt had. One of Rembrandt's etchings is shown in figure 1.1.



Figure 1.1 The amazing detail of Rembrandt's etchings. Adverse Fortune, 1633

HTML was originally developed as a language to render research documents. HTML was never meant to be used to create the web applications of today, and if someone from back in

the time of the early Web was to see a modern web application, I don't think they would believe that it was created using a descendant of the original HTML. There have been some amazing sites developed using HTML, CSS, and JavaScript. I consider the developers behind these sites to be like modern Rembrandts, taking technologies that were never meant to do these amazing things and stretching them to their limits. Unfortunately, there are not many Rembrandts in the world, and you can't count on having one on your team. And even if you had one, it would still be more time consuming and painstaking to develop a rich application using these technologies.

By using technologies that were designed from the ground up to provide interactive experiences, including multimedia, animations, and vector graphics, you don't necessarily need a Rembrandt, but if you have one, they can take an RIA application and take it to its limits and beyond. Now let's take a look at some of the features of RIAs and how they help to provide a better user experience.

### **VECTOR GRAPHICS**

Computer graphics typically fall into two major categories. The most common format is raster graphics, also known as bitmap graphics. This group includes photos, icons, and other files where each pixel is represented in the stored file. Many of these files are compressed to save space, sometimes using what is called "lossy" compression, which reduces the size of the file, but some detail is lost in the image. The most common image format, JPEG, uses lossy compression. This is usually fine for photos, since the human eye can't easily perceive the loss of detail in an image as varied in color and brightness as a photo is. If you were to put some text and draw it on a solid background and then save the image as a JPEG, you would see some pixels around the edges of the text which are slightly off color. These variations are called artifacts, and are a result of the compression process.



Figure 1.2 The result of lossy compression on a simple image. Notice the imperfections around the edges of the text in the second image.

So if we want to use an image to display text, or a bar graph, or any other graphic that has sharp edges large portions of a solid color, a lossy image won't give us the results we're looking for. There are other file formats, which are compressed with lossless compression, that can provide crisp, sharp graphics but the files are much larger, and are not uniformly supported on the most common browsers.

The other major graphics category is vector graphics. In vector graphics, the data is stored as a collection of paths with supporting data that describes the characteristics of the path, such as color, thickness, whether the path is closed, which would result in a shape, and if so, what color to fill the shape with. There are many benefits to vector graphics. First of all, since only the path information is stored instead of every pixel's value, the graphic can typically be represented with less data, sometimes much less. Vector graphics are not typically suitable for

storing a photo for some of the same reasons that lossy compression is well suited for photos. The color variations in a typical image would make it prohibitive to use a series of path data to store the graphics information.

What vector graphics are very good at is representing shapes and text. Anything that can be described as a series of lines and curves to outline a shape can easily be stored as a vector graphic. Another benefit of vector graphics is that they are resolution independent. This means that if you want to display your vector graphic at ten times its size, it's just a simple matter of multiplying the end points and control points of the path by a scale factor of ten and then the vector graphic can be drawn at that size. If you were to scale a raster graphic by the same amount, the level of detail would not increase, and the edges would look jagged.

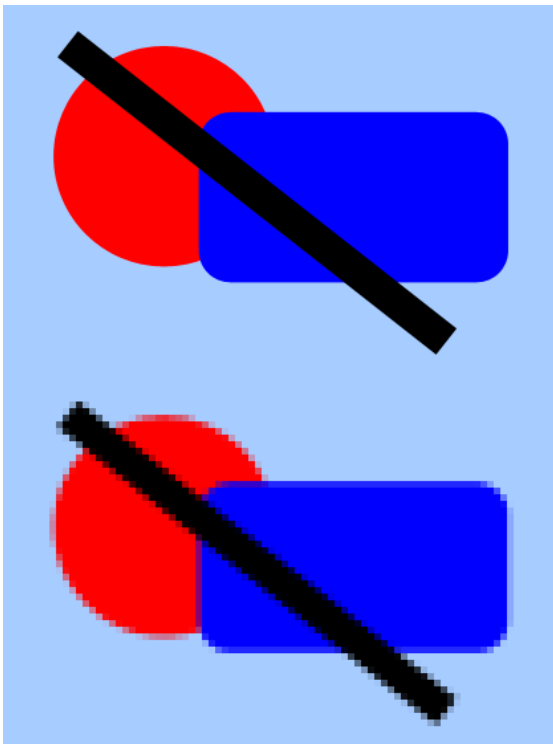


Figure 1.3 The impact of scaling on vector and raster graphics. The first image is a vector graphic scaled by 500 percent, while the second is the same graphic converted to a raster image and then scaled by 500 percent.

This ability to easily resize vector graphics is why most fonts are stored in this format. Take any font in a word processing program and blow it up to 100 point text and it will still

have smooth edges. Along with scaling, vector graphics can also be manipulated in other ways such as stretching, skewing and rotating without any loss of detail.

Vector graphics have been around for a long time. Some of the earliest arcade games such as Asteroids and Battlezone used this technique. However, until RIA technologies appeared on the scene, there was no way to display vector graphics in all of the major browsers. This has caused web pages to increasingly use raster images to reach the desired appearance, significantly increasing the amount of data that needs to be downloaded to display a typical web page. RIAs were built from the ground up to natively support vector graphics and reap the benefits that they give you.

### **BREAKING AWAY FROM THE BOXES**

In HTML, all controls are rectangular and oriented in the same direction. With the introduction of Cascading Style Sheets (CSS) these boxes could be positioned anywhere in the window, even overlaying other boxes, but there was still the limitation that these boxes be rectangular and not rotated. Web designers have used various methods to make their pages look less boxy, usually using some cleverly constructed bitmap images. Let's say, however, that we want a video to play stretched and skewed to give it a three dimensional effect and we want to round the corners to make it look like it's playing on an old television. This is something that is impossible to achieve using traditional web technologies.

Another thing you may notice with traditional web applications is that whenever a form is displayed with text boxes, check boxes, and radio buttons, they always look the same, and the web designer is limited in what they can do to make these controls fit in with the rest of the design of the page. Because of the strong vector graphic support and the ability to easily create composite controls that combine shapes, text, and images, RIA applications have much more flexibility when it comes to the look and feel of input controls and they can be styled to enhance the user experience instead of just looking like something that is slapped on top.

As designers attempt to make better use of page real estate, they are turning to techniques where they can overlay information and controls on top of each other. A good example of this is a text box where before any text is entered, a hint is displayed over the text box explaining the information that the text box expects. Once the first character is entered, this hint disappears. This is a much better use of space than having a separate block of text to the side which would display this same information. If you're running Windows Vista and do a local file search, you will see that the bread crumb showing your current path doubles as a progress bar for the search. Another technique that is used often in Vista to maximize screen real estate is to display objects as semi-transparent, and when the mouse is positioned over these objects, they can fade into view. This is something that would be difficult or impossible to achieve using traditional web technologies but is trivial in an RIA application. These types of effects are done using animations. Let's take a look at the animation capabilities of RIAs in a bit more detail.

### **ANIMATIONS**

An animation in RIA terms is simply the changing of the value of a property or properties over a period of time. The various RIA frameworks name these animations different things, Flash,

for example, calls them timelines, and Silverlight calls them storyboards. In Flash, these animations are frame based, where there is a default of 12 frames per second, and you specify how many frames it will take for a certain animation to execute. In Silverlight, animations are time based, but can contain multiple key frame times where the target value can change.

Some basic animation has been done for years in traditional web pages, typically using JavaScript and timers to modify the value of a document property over a given time. With the advent of some sophisticated AJAX libraries, some of this animation has become easier, but there are still browser differences to deal with, and animations still typically have to be written in code. With Silverlight and other RIA technologies these animations can be specified in the design of the layout of the page, and simply kicked off when needed, based on a mouse entering or leaving a control, or a keyboard input or mouse click, or other external inputs. So what types of properties can be controlled via an animation? Well clearly we need to be able to change the position and size of an object, but animations can also change the color of an object, its level of transparency, the rotation angle, the stretch or skew of an object, or pretty much any other property you can think of that can be represented as a number or a collection of numbers. Animations are very powerful in Silverlight, and we'll cover in depth later how to get the most out of this feature.

#### **CLIENT SIDE STORAGE**

Web applications can't have free access to the local file system for obvious reasons. If this access was granted, then sensitive information could be retrieved from the local computer and sent to the server. Also critical system files could be overwritten, exposing the user to malware or worse. Much of the effort in web security over the past few years has been to protect the casual user from unseen threats. Normally the only way a web application can store information on the user's computer and retrieve that information later is through the use of cookies. These cookies can store small amounts of information, typically less than 4 kilobytes.

Let's say you had a web application where a user could enter a blog post. This blog post could be hundreds of words or more, and is entered via a web form. Now let's say one of your users is almost done with the post, and they lose their internet connection, or the browser crashes, or they accidentally close the page. All of that information is lost since the data is not sent to the server until they hit the Submit button. This is one case where it would be very useful to be able to save a working set of the data to the local file system periodically until the data is successfully uploaded.

To provide some access to the local file system to store data while protecting the user's sensitive data, a technique known as sandboxing or protected storage is used which allows the application to write to and read from a folder which only that application can access. This space is not unlimited, and defaults to 100 kilobytes per application in Flash, or 1 megabyte in Silverlight.

If all of these RIA platforms offer these benefits, why would you choose one over the other? Each option implements these features using different languages, and development is done using different tools. The next section highlights some of these differences to assist in deciding which is right for you.

### **1.1.2 A quick look at alternative RIA technologies**

I'm a strong believer in using the right technology for the right task. By having at least some basic knowledge of the various technologies that are available, an educated decision can be made about which technology to use. Too many times a technology is chosen because it's "cool" or "trendy". A few years ago, a company started a project where a very functional and stable application was rewritten in a new language because the lead architect had read that code using this language was easier to maintain and more productive to develop in. The problem was that the development team had no experience in working with this language, and wrote code in it as they would have in the languages that they were familiar with. The project went way beyond schedule and over budget, under-delivered on features, and could not match the performance of the original application. The language was C++, which obviously had tremendous success and revolutionized application development, but it was the wrong choice for that project given the resources available.

I won't sit here and bash competing RIA technologies, and each of them will have situations where it is the best choice. The best technology for the job depends on the task to be completed, the audience for the application, the development resources available, the prerequisites required to run a particular application, and many other factors. So let's take an objective look at some of the positive and negative aspects of a few of the more popular RIA technologies so that a more educated decision can be made on which to use. The most familiar and well established of these are Flash and Ajax, but newcomers like OpenLaszlo and Silverlight are gaining ground quickly.

#### **AJAX**

There is some debate as to whether Ajax is an RIA technology. Ajax has some of the characteristics associated with Rich Internet Applications, but not the entire set of features that classify an application as an RIA. Ajax stands for Asynchronous JavaScript and XML, and is more of a concept or a technique than a standard. It takes advantage of the XMLHttpRequest interface to make requests to the server and retrieve data without posting back and refreshing the entire page. It incorporates earlier concepts like Dynamic HTML and the browser's Document Object Model.

For browser based applications, Ajax was a major step in the right direction, providing a framework for developing dynamic web applications by avoiding some of the inherent limitations in HTML based applications. However, since Ajax is built on top of HTML and Javascript, some of the limitations still exist. Each browser renders HTML a bit differently, and there are differences in how the document object model is implemented. JavaScript is not a strongly typed language, and is interpreted instead of compiled. It is also difficult to take advantage of object oriented techniques in JavaScript.

Any large Ajax based application requires a tremendous amount of JavaScript. This can lead to several issues. It can take a long time to download all of the JavaScript required for the application. JavaScript can suffer from performance issues on complex tasks. If a page has

more than one Ajax control on the page, especially if using different Ajax based frameworks, the developer needs to be careful to fully understand the interaction between these libraries. Finally, based on the sheer number of lines of JavaScript code, the application can become difficult to maintain.

Those that do not consider Ajax to be an RIA platform point to the lack of advanced graphics and multimedia capabilities. There have been some advances in Ajax technology recently to allow it to do more vector style graphics and if these advances continue, the line will probably be blurred more between the features available in these technologies. Let's move on and take a look at some platforms that fall squarely into the RIA camp.

### **ADOBE FLASH AND FLEX**

When Flash was first introduced in 1997, it was simply a technology for drawing and animating vector based graphics. It has evolved significantly over the years, adding support for multimedia, a scripting language called ActionScript, and various improvements to usability. Recent surveys show that Flash is installed on over 97 percent of internet enabled desktop computers in mature markets. Because of this huge install base, Flash has become the tool of choice for web content that would be difficult to present using traditional web technologies.

Adobe Flex is built on top of Flash and adds the ability to design graphic user interfaces using an XML-based language called MXML. This allows for a separation of design from code in the same way that XAML provides for WPF and Silverlight. Adobe considers Flex to be their flagship RIA platform. It is targeted more at developers where Flash is more of a designer tool.

Flash is a mature technology. It has been around for over ten years and that has its benefits, but also can be a problem. As new web technologies have gained prominence, such as XML, Web Services, and Ajax, the ability to work with these technologies has been added to Flash. The functionality is there, but it may not be as easy to use or tightly integrated as it would be if Flash was designed to use those technologies from the beginning.

Since Flash and Flex use ActionScript for their code, developers need to be proficient in a language that is limited to use within the confines of Adobe technologies. Adobe is extending their reach to the desktop, with their AIR offering, which allows for desktop applications to be written using ActionScript, Flash and Flex. If you have a team of ActionScript developers, then it may be difficult to come up with a compelling argument at this point in time as to why Silverlight would be a better choice. Unfortunately, ActionScript developers are not nearly as common as developers proficient in Microsoft .Net based languages, so it may be difficult to find the resources needed to develop the application.

### **OPENLASZLO**

OpenLaszlo is a server technology which generates rich client applications that can run as either Flash controls or Dynamic HTML. Applications are written using LZX, an XML based format with embedded JavaScript. Since LZX is an intermediate format, and is compiled into the client format before it is deployed, it is possible and that Silverlight could be added to the list of client side technologies that can run Laszlo applications.

The ability to deploy as Dynamic HTML is one of Laszlo's major selling points, since no extra plug-ins are needed for the browser to run the client application. So if your requirements

are that your application can run in any browser without requiring the user to install any additional software, then OpenLaszlo may be the right option for you. When running in DHTML mode, a Laszlo application requires a Java component on the Server. If the application is deployed as a Flash application, it can be compiled to Flash and deployed from any server, like a typical Flash application.

Flash and Flex advocates argue that a vast majority of browsers already have the Flash plug-in installed, and Flex applications are easier to write. Both Silverlight and Flex have the advantage of separation of code and the markup used for layout which can make the code more manageable. Developing using LZX is more like traditional ASP development where the logic and the layout are in the same file. For small projects, this may not be a problem, but as the application grows, it can get out of control quickly.

LZX is another language where it may be hard to find developers proficient in it and most likely more difficult than finding ActionScript developers.

#### **SILVERLIGHT**

Silverlight shares many of the features of the other RIA technologies. There is sandboxed offline storage, a markup language for specifying user interface layouts, and powerful multimedia and graphics capabilities.

There are heated debates all over the forums arguing the benefits of Silverlight and other RIA platforms, and which technology to choose when starting a new RIA project. Personally, I prefer Silverlight because I can use the tools that I am comfortable with, like Visual Studio, using my favorite language, C#. The other technologies have some benefits as well, for example there are some features that are available in Flash or Flex that are not currently supported in Silverlight. Since Silverlight is such a new technology, the development team had to determine what the top priority features were, and implement those first. As Silverlight matures, it will need to provide the missing features to keep pace with other RIA technologies.

Being the new kid on the block has its advantages. Silverlight can learn from the mistakes of the past, and take full advantage of the latest advances in web technologies because it was built from the ground up with them in mind.

Silverlight 2 is a major step toward the convergence of desktop and web development. By providing a micro version of the CLR, developers can use the same tools and techniques whether they are developing for Windows or the web. The Silverlight CLR was built from the ground up to be a high performance runtime and is completely compatible with the full .Net Framework. Code that runs on top of the CLR is referred to as managed code and the key benefit of managed code is that memory is freed automatically when the object associated with that memory is no longer needed. This is done via a process called garbage collection. The garbage collector in Silverlight 2 is identical to the full .Net version. Compare this to the Compact Framework that runs on mobile devices. The Compact Framework has its own garbage collector which is primitive in comparison to the garbage collector available in the .Net Framework. The Silverlight CLR is scaled down in scope, only implementing the classes critical to developing a Silverlight application, but what is there is high performance, robust, and built upon years of managed code experience.

Okay, so maybe I'm not being completely objective when discussing the benefits of using Silverlight. After all, this is a Silverlight book, and I can't hide the fact that I love the technology. Let's get started and see how easy it is to create your first Silverlight application.

## **1.2 The tools of the trade**

They say that a craftsman is only as good as his tools. Fortunately, when building Silverlight applications, the tools are the some of the best available. Microsoft Visual Studio 2008 is the key developer tool for Silverlight 2. Along with the developer tools, Microsoft has also created Expression Blend, a tool for graphics designers which offers an unprecedented level of collaboration with the development team.

### **MICROSOFT VISUAL STUDIO 2008**

Visual Studio is the flagship development platform for Microsoft technologies. If you've never used Visual Studio before, it is arguably the most advanced Integrated Development Environment (IDE) available today. I started using Visual Studio back in 2001 when Microsoft released Visual Studio 7. The granddaddy of the modern Visual Studio was released in 2002 and coincided with the first release of the .Net Framework. The Visual Studio experience has evolved over the years, and each release brings features that improve the developer experience.

To use Visual Studio 2008 for Silverlight development, you will also need to download and install the Microsoft Silverlight Tools for Visual Studio 2008 which includes the project templates and supporting files for creating Silverlight applications with Visual Studio.

Visual Studio 2008 is probably the most significant release since 2002. In the three years since the previous release, technology has moved forward significantly, and with these advances come some new issues, requiring new features. The outstanding debugger and IntelliSense have been improved and extended, including IntelliSense for XAML documents. And if you need to work with JavaScript for a tight integration between your Silverlight controls and the surrounding browser page, you'll be thrilled with the improved debugging and IntelliSense for JavaScript.

If you don't have access to the full versions of Visual Studio 2008, you can develop Silverlight applications using the free Visual Web Developer 2008 Express Edition. The Express edition is a full development environment which is more than adequate for most development projects. It includes many of the features of the full Visual Studio editions and is a great alternative.

### **MICROSOFT EXPRESSION BLEND**

Microsoft Expression Blend is a revolutionary design tool. It was built from the ground up using WPF technology to create an environment that designers are familiar with. It behaves the way designers expect it to. What makes it unique is that it understands the Visual Studio project format, and allows designers to work on the same exact files and projects as the developers.

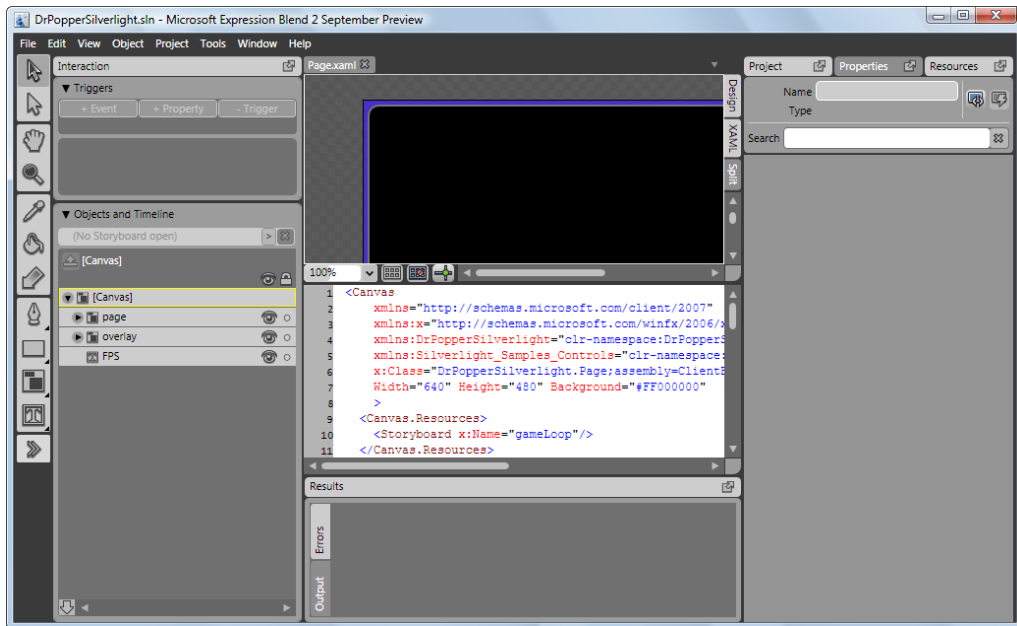


Figure 1.4 Microsoft Expression Blend is targeted at how graphics designers work.

In the past, designers would create the application's look and feel in a tool like Photoshop or Illustrator, and then hand it over the wall to the developers, who would do their best to make the application look like what the designer had envisioned. In a best case scenario, the developer would slice up the graphics and come up with a reasonable approximation of what the designer had in mind. Things rarely would go that smoothly. It was more common for the developers to push back and say that the technology being used to create the application could not faithfully implement the experience that the designer had in mind. In even worse cases, developers would design an application without getting a designer involved early, and then near the end of development, a designer would need to get involved and attempt to pretty up the application as best as they could.

With the tight integration between Visual Studio and Blend, with designers and developers working off of the same project, teams can develop applications in any way that works best for them. They can do a design first approach, or prototype first, or design and develop concurrently. And even if most of the application is done before the designer gets involved, all is not lost, since as long as the designer follows the same naming conventions, the whole front end could be swapped out fairly painlessly.

Expression Blend provides a visual design surface allowing designers to combine vector graphic primitives, gradient fills, imported XAML content, and other media to create user interfaces. It also provides robust support for creating animation timelines.

Anything that can be done with Blend can be done with other tools, or in a text editor. Blend can be a great learning tool. By studying the XAML that it produces, you can learn new tricks and techniques for your own XAML. A free 60 day trial of Blend is available for download from [microsoft.com](http://microsoft.com). For more information on using Blend, please refer to Appendix A: Getting Started with Microsoft Expression Blend.

### 1.3 The world's simplest Silverlight application

So let's get started and see Silverlight in action. Open Visual Studio 2008 and select File->New->Project, and in the C# section, select Silverlight. You should see a project type of "Silverlight Application" as shown in figure 1.5.

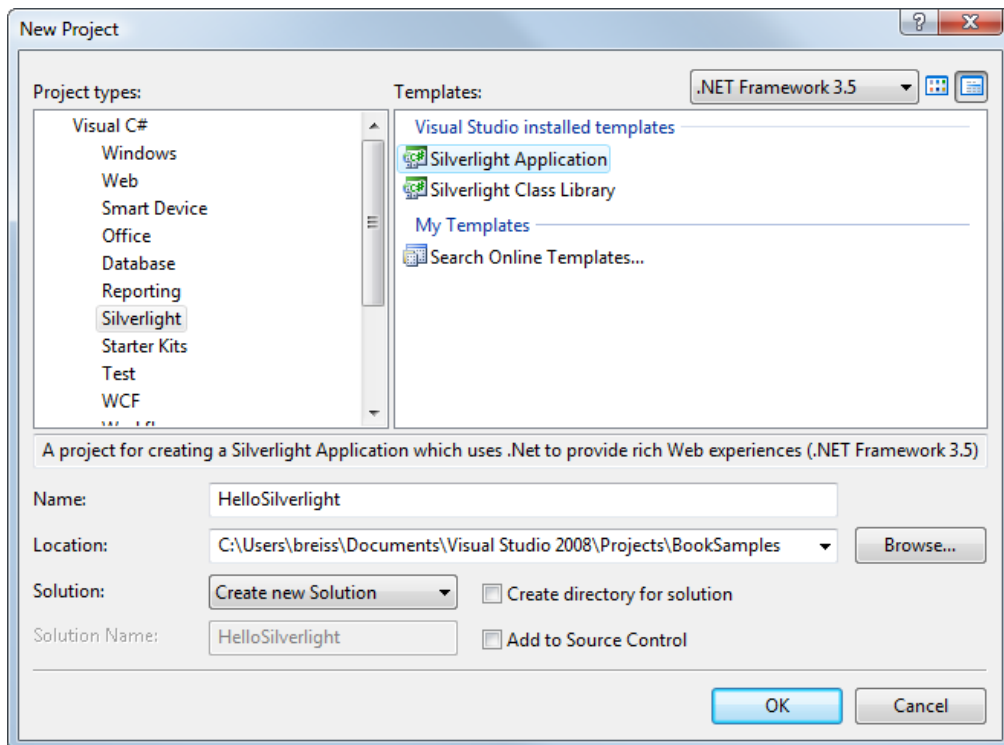


Figure 1.5 The Silverlight section of the Visual Studio 2008 New Project dialog

Select this and specify a project name of `HelloSilverlight` and click OK. This kicks off the New Silverlight Application wizard. Now we're given some choices. We can either create a separate web project to host the Silverlight application, or we can generate an HTML test page in the Silverlight project to host our Silverlight control. If you want to interact with the web server from your Silverlight control, or combine ASP.NET and Silverlight content on the page,

then you will want to create a web project in your solution. This, however, is the world's simplest Silverlight application, so let's keep it as simple as possible and choose to generate an HTML test page in our project as shown in figure 1.6.

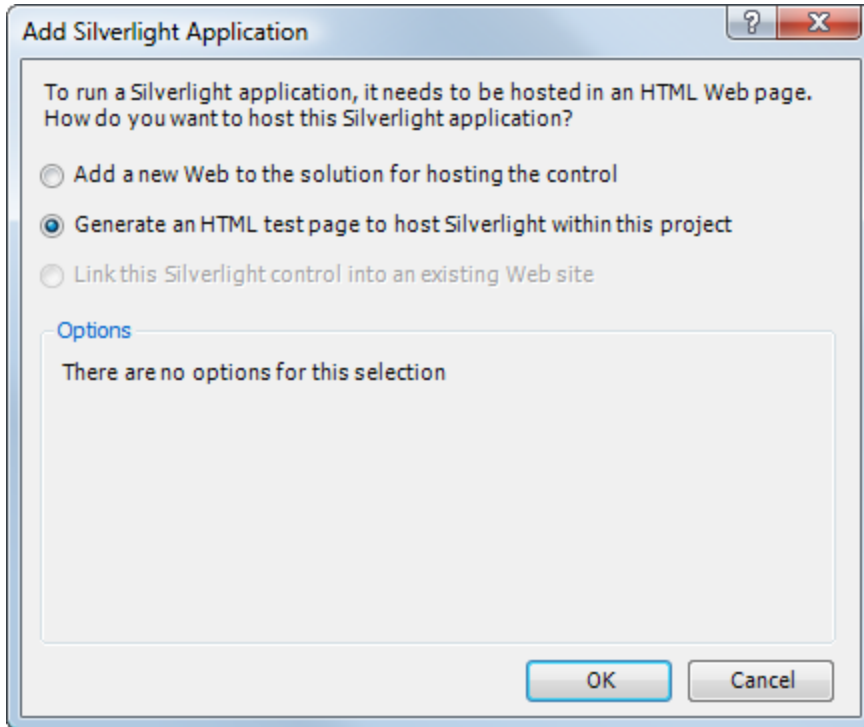


Figure 1.6 The Add Silverlight Application wizard dialog

Now if you click OK, Visual Studio will create the Silverlight application for you. Let's take a look at some of the files that are generated as part of a Silverlight project.

Table 1.1 The files created by the C# New Silverlight Application Wizard

File	Description
Page.xaml	XAML file which contains the markup for the Page object. The Page object contains the root panel of the control.
Page.xaml.cs	Code behind file for the Page object .

App.xaml	Global resources go here.
App.xaml.cs	Contains code that runs when Silverlight is loaded, including loading the Page object.

Go ahead and run the project and see what happens. It looks like an empty browser window, right? Actually it isn't, it's just that the Silverlight control has a white background and so does the web page. You can confirm this by right clicking in the browser window, and you'll get the Silverlight context menu. Let's take a closer look at the Page.xaml file and see what we can do to make it visible.

### HEY, THAT LOOKS LIKE XML

The Page.xaml file is the root of your Silverlight control. It is roughly equivalent to a default.aspx page in ASP.NET or a Form1.cs in Windows forms programming. Any graphical element that you add to the control will be a child or other descendant of this root. To view the markup for Page.xaml, go to the Solution Explorer tab, and double click on the Page.xaml file.

#### Listing 1.1 Page.xaml file which is generated by the new project wizard

```
<UserControl x:Class="HelloSilverlight.Page"
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="400" Height="300">
  <Grid x:Name="LayoutRoot" Background="White">
    </Grid>
</UserControl>
```

#### A Background is White

If you've ever done any XML programming, the above should look pretty familiar. XAML (pronounced *Zammel*) stands for eXtensible Application Markup Language and is based on XML. It follows standard XML such as namespaces and schemas to create well formed documents. It has become Microsoft's standard format for laying out user interfaces, and has many benefits over previous techniques.

In traditional Windows Forms development, the designers built into Visual Studio would generate code in the language of whatever the application was being coded in. This meant that additional logic would need to be added to the designers so that it would know how to generate C# or VB.Net code. Every time a new language was added, new logic would need to be implemented to generate the appropriate code. This tight coupling of the designer to the language that was being used severely limited the tools available to design Windows Forms applications.

The other major benefit of XAML is that separating the code from the interface is what allows us to realize the benefits stated earlier about having designers and developers work off of the same project. Do you really want your designers changing code? It's useful for designers to know a bit about the application technology, but it's too much to expect most designers to

have enough knowledge to change the code without unexpected side effects. In XAML, as long as the designer and the developer agree on some basic naming conventions, the designer could completely rewrite the user interface, even using different graphical element types, as long as the names are consistent.

Now back to our first Silverlight application. In order to see the Silverlight control, let's change the Background property to Blue:

```
<Grid x:Name="LayoutRoot" Background="Blue">
```

Go ahead and run the application again. You should now see a blue box on a white background.

Now let's add some text. You can add text to a Silverlight application with the TextBlock element. The TextBlock element's attributes are modeled after the style attributes in CSS and HTML. In general, Silverlight controls follow this convention, reducing the amount of new information that needs to be absorbed when getting used to Silverlight. Let's make the TextBlock say "Hello, Silverlight!" and give it a foreground color of White and a font size of 45. The Page.xaml after these changes is shown in listing 1.2.

#### Listing 1.2 Page.xaml with a TextBlock

```
<UserControl x:Class="HelloSilverlight.Page"
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="400" Height="300">
  <Grid x:Name="LayoutRoot" Background="Blue">
    <TextBlock
      Text="Hello, Silverlight!"
      Foreground="White"
      FontSize="45" />
  </Grid>
</UserControl>
```

Now if you run the application, it will look like figure 1.7.



Figure 1.7 Our simple Silverlight application with text

So that's all there is to it, you've written your first Silverlight application! Now let's see what we have learned so far.

## ***1.4 Summary***

RIAs such as Silverlight are convergent technologies and a stepping stone toward being able to write once and run anywhere. As RIAs evolve, the line between desktop applications and web application will be harder to discern, and hopefully in the near future we will just be talking about applications and not desktop or web applications. Depending on your vendor of choice, some RIA technologies are probably more attractive than others, with each staking a claim to why it should be the technology for your next project. Silverlight 2 has a compelling story for Microsoft developers since they can use their language and tools of choice. It is also attractive to developers that are currently using competing technologies since the skills they learn while becoming proficient in Silverlight will also serve them well for developing other applications using Microsoft technologies.

Aiding in the adoption of Silverlight 2 is a high performance runtime which cannot be matched by the competing technologies and some of the best tools in the business to make Silverlight developers extremely productive. Developers and designers can work off of the same projects and code base, allowing for collaboration that has not previously been possible. Add to this the ability to program your application with your language of choice, and Silverlight stands apart from all other alternatives in its ability to create fast, robust, and compelling web

applications quickly and easily. Throughout the rest of the book, we'll explore the key features of Silverlight and we'll develop some real world samples along the way. The next chapter covers the fundamentals of XAML in Silverlight and how to put them to use.