Introducing Microsoft® Silverlight™ 2, Second Edition

Laurence Moroney

To learn more about this book, visit Microsoft Learning at
http://www.microsoft.com/MSPress/books/12086.aspx

9780735625280

Table of Contents

Part I Introducing Silverlight 2

5 Transformations 109

6 The Silverlight Browser Control 133

Part II Programming Silverlight 2

11 Media, Ink, and Deep Zoom 247

Chapter 1

Introducing Silverlight 2

Silverlight represents the next step toward enriching the user's experience through the technology of the Web. The goal of Silverlight is to bring the same fidelity and quality found in the user interfaces (UIs) associated with desktop applications to Web applications, allowing Web developers and designers to build applications for their clients' specific needs. It is designed to bridge the technology gap between designers and developers by giving them a common format in which to work. This format will be rendered by the browser and will be based on XML, making it easy to template and to generate automatically. The format is XAML—Extensible Application Markup Language.

Before XAML, a Web experience designer would use one set of tools to express a design using familiar technology. The developer would then take what the designer provided and would interpret it using the technology of his or her choice. The design would not necessarily transfer properly or problem-free into development, and the developer would need to make many alterations that could compromise the design. With Silverlight, the designer can use tools that express a design as XAML, and the developer can pick up this XAML, activate it with code, and deploy it.

Microsoft Silverlight is a cross-browser, cross-platform plug-in that was developed to deliver rich media experience and rich interactive Internet applications via the Web. It offers a full programming model that supports AJAX, .NET, and dynamic languages such as Python and Ruby. Silverlight 1.0 is programmable by using actual Web technologies including AJAX, JavaScript, and DHTML. Silverlight 2 adds dynamic and .NET language support, as well as a host of new features that are only possible when using the .NET Framework, such as Isolated Storage, Networking, a rich control set, and more.

The first part of this book will introduce you to the fundamentals of Silverlight 2 by looking at the design and development tools that are available to you, and the second part will examine the programming model more closely.

Silverlight and User Experience

Silverlight is designed to be part of a much larger ecosystem that is used to deliver the best possible end-user experience. There are a number of typical scenarios for accessing information via the Internet:

- Mobile devices
- Digital home products

- Unenhanced browsers (no plug-ins)
- Enhanced browsers (using plug-ins such as Flash, Java, or Silverlight)
- Desktop applications
- Office productivity software

Over the years, users' expectations about how these applications should work have evolved. For example, the *expectation* is that the experience of using an application on a desktop computer should provide more to the user than the same type of application on a mobile device because, as users, we are accustomed to having much more power on the desktop than we do on a mobile device. In addition, many users assume that "because this application is on the Web," it may not have the same capacity level as a similar desktop application. For example, a user may have lower expectations about a Web-based e-mail application because they don't believe it can offer the same e-mail capability that office productivity software such as Microsoft Office Outlook provides.

However, as these platforms are converging, the user's expectations are also increasing—and the term *rich* is now commonly used to describe an experience above the current baseline level of expectation. For example, the term "rich Internet application" was coined in response to the increased level of sophistication that Web users were seeing in applications powered by AJAX to provide a more dynamic experience in scenarios, such as e-mail and mapping. This evolution in expectations has led to customers who now demand ever richer experiences that not only meet the needs of the application in terms of functionality and effectiveness but also address the perception of satisfaction that the user has with a company's products and services. This can lead to a lasting relationship between the user and the company.

As a result, Microsoft has committed to the User Experience (UX) and is shipping the tools and technologies that you as a developer can use to implement rich UX applications. Additionally, they are designed to be coherent—that is, skills in developing UX-focused applications will transfer across the domains of desktop and Web application development. So, if you are building a rich desktop application but need a Web version, then you will have a lot of cross-pollination between the two. Similarly, if you are building a mobile application and need an Internet version, you won't need two sets of skills, two sets of tools, and two sets of developers.

Concentrating on the Web, Figure 1-1 shows the presentation and programming models that are available today. As you can see, the typical browser-based development technologies are CSS/DHTML in the presentation model and JavaScript/AJAX/ASP.NET in the development model. On the desktop, with the .NET Framework 3.x, XAML provides the presentation model, and the framework itself provides the development model. There is an overlap between these, and this is where the Silverlight-enhanced browser provides a "best of both worlds" approach.

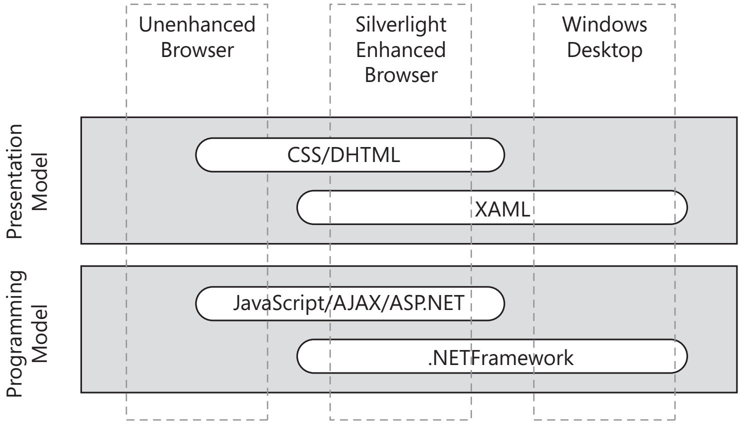


FIGURE 1-1 Programming and presentation models for the Web.

The typical rich interactive application is based on technologies that exist in the unenhanced browser category. The typical desktop application is at the other end of the spectrum, using unrelated technologies. The opportunity to bring these together into a rich application that is lightweight and runs in the browser is realized through the Silverlight-enhanced browser that provides the CSS/DHTML and XAML design model and the JavaScript/AJAX/.NET Framework programming model.

Silverlight achieves this by providing a browser plug-in that enhances the functionality of the browser with the typical technologies that provide rich UIs, such as timeline-based animation, vector graphics, and audiovisual media. These are enabled by the Silverlight browser-based XAML rendering engine. The rich UI may be designed as XAML, and because XAML is an XML-based language and because XML is just text, the application is firewall-compatible and (potentially) search-engine friendly. The browser receives the XAML and renders it.

When combined with technology such as AJAX and JavaScript, this can be a dynamic process—you can download snippets of XAML and add them into your UI, or you can edit, rearrange, or remove XAML that is currently in the render tree using simple JavaScript programming.

Silverlight Architecture

As I mentioned, the core functionality of Silverlight is provided by a browser plug-in that renders XAML and provides a programming model that can be either JavaScript and browser-based or the .NET Framework and CLR-based. The architecture that supports this is shown in Figure 1-2. When scripting the control in the browser, the main programming interface that is exposed in Silverlight 1.0 is via the JavaScript DOM API. This allows you to catch user events that are raised within the application (such as mouse moves or clicks over a specific element) and have code to execute in response to them. You can call methods on the JavaScript DOM for XAML elements in order to manipulate them—allowing, for example, control of media playback or animations to be triggered.

For a richer and more powerful experience, you can also program an application that is rendered by the control using the new .NET Framework CLR. In addition to what you can do in JavaScript, this capability offers many of the namespaces and controls that come as part of the .NET Framework, allowing you to do things that are either very difficult—or not possible—in JavaScript, such as accessing data with ADO.NET and LINQ, communicating with Web Services, building and using custom controls, and so on.

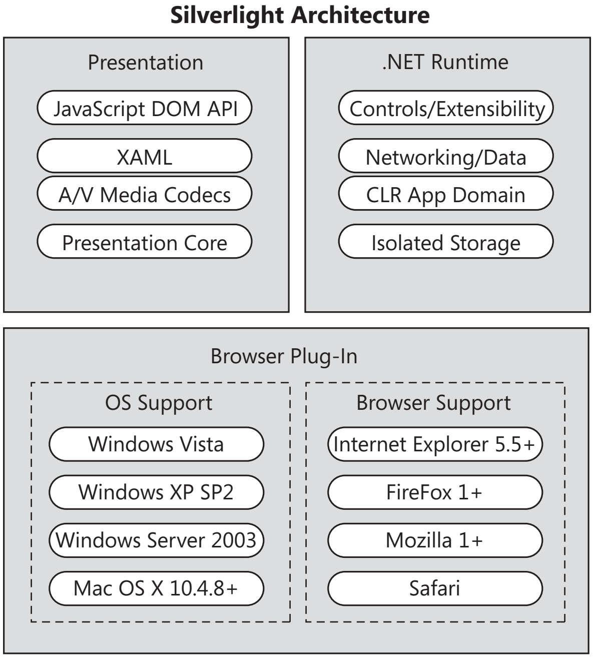


FIGURE 1-2 Silverlight architecture.

Additionally, the presentation runtime ships with the software necessary to allow technologies such as WMV, WMA, and MP3 to be played back in the browser *without* any external dependencies. So, for example, Macintosh users do not need Windows Media Player to play back WMV content—Silverlight is enough. Underpinning the entire presentation runtime is the presentation code, and this manages the overall rendering process. This is all built into the browser plug-in that is designed to support the major browsers available for both Windows and the Macintosh.

The architecture of a simple application running in the browser using Silverlight is shown in Figure 1-3.

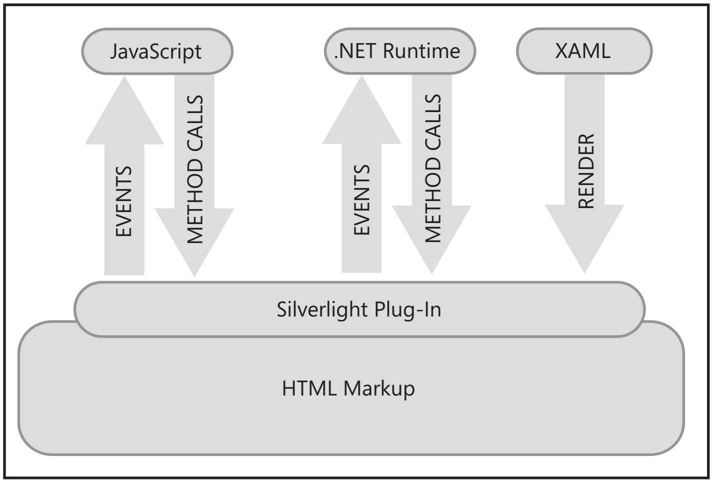


FIGURE 1-3 Application architecture with Silverlight.

As the application runs within the browser, it is typically made up of HTML. This markup contains the calls to instantiate the Silverlight plug-in. As users interact with the Silverlight application, they raise events that can be captured by either JavaScript or.NET Framework functions. In turn, program code can make method calls against the elements within the Silverlight content to manipulate it, add new content, or remove existing content. Finally, XAML can be read by the plug-in and rendered. The XAML itself can exist inline in the page, externally as a static file, or as dynamic XAML returned from a server.

Silverlight and XAML

Now that we've taken a high-level look at the architecture of Silverlight and how a typical application will look, let's examine the base technology that holds the UX together: XAML.

XAML is an XML-based language that is used to define the visual assets of your application. This includes UIs, graphical assets, animations, media, controls, and more. It was introduced by Microsoft for the Windows Presentation Foundation (WPF), formerly Avalon, which is a desktop-oriented technology and part of the .NET Framework 3.0 and beyond. It's designed, as discussed earlier, to bridge the gap between designers and developers when creating applications.

The XAML used in Silverlight differs from that in the WPF in that it is a *subset* that is focused on Web-oriented features. So, if you're familiar with XAML from the WPF, you'll notice some missing tags and functionality, such as the *<Window>* element.

XAML uses XML to define the UI using XML elements. At the root of every Silverlight XAML document is a container element, such as a *Canvas*, that defines the space on which your UI will be drawn. When building a Silverlight Web application, you'll have a root *Canvas* that contains the XML namespace declarations that Silverlight requires.

Here's an example:

```
<Canvas
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="640" Height="480"
  Background="White"
  >
</Canvas>
```

You will notice that two namespaces are declared. The typical XAML document contains a base set of elements and attributes as well as an extended set, which typically uses the *x:* prefix. An example of an extended namespace attribute is the commonly used *x:Name*, which is used to provide a name for a XAML element, allowing you to reference it in your code. The root *Canvas* element declares the namespace location for each of these.

The *Canvas* element is a container. This means that it can contain other elements as children. These elements can themselves be containers for other elements, defining a UI as an XML document tree. So, for example, the following is a simple XAML document containing a *Canvas* that contains a number of children, some of which are *Canvas* containers themselves:

```
<Canvas
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="640" Height="480"
  Background="Black"
  >
    <Rectangle Fill="#FFFFFFFF" Stroke="#FF000000"
        Width="136" Height="80"
        Canvas.Left="120" Canvas.Top="240"/>
    <Canvas>
        <Rectangle Fill="#FFFFFFFF" Stroke="#FF000000"
                Width="104" Height="96"
                Canvas.Left="400" Canvas.Top="320"/>
        <Canvas Width="320" Height="104"
                Canvas.Left="96" Canvas.Top="64">
            <Rectangle Fill="#FFFFFFFF" Stroke="#FF000000"
                          Width="120" Height="96"/>
            <Rectangle Fill="#FFFFFFFF" Stroke="#FF000000"
                          Width="168" Height="96"
                          Canvas.Left="152" Canvas.Top="8"/>
      </Canvas>
   </Canvas>
</Canvas>
```

Here you can see that the root *Canvas* has two children, a *Rectangle* and another *Canvas*. This second *Canvas* also contains a *Rectangle* and a *Canvas*, and the final *Canvas* contains two more *Rectangle*s. This hierarchical structure allows for controls to be grouped together logically and to share common layout and other behaviors.

Silverlight XAML supports a number of shapes that can be combined together to form more complex objects. You'll find a lot more details about using XAML in Chapter 4, "XAML Basics," but a few of the basic shapes available include the following:

- ***Rectangle*** Allows you to define a rectangular shape on the screen
- ***Ellipse*** Allows you to define an ellipse or circle
- ***Line*** Draws a line connecting two points
- ***Polygon*** Draws a many-sided shape
- ***Polyline*** Draws many line segments
- ***Path*** Allows you to create a nonlinear path (like a scribble)

In addition, XAML supports *brushes*, which define how an object is painted on the screen. The inside area of an object is painted using a *fill* brush, and the outline of an object is drawn using a *stroke*. Brushes come in many types, including solid color, gradient, image, and video.

Following is an example using a *SolidColorBrush* to fill an ellipse:

```
<Ellipse Canvas.Top="10" Canvas.Left="24"
          Width="200" Height="150">
      <Ellipse.Fill>
         <SolidColorBrush Color="Black" />
      </Ellipse.Fill>
</Ellipse>
```

In this case, the brush uses one of the 141 Silverlight-supported named colors, *Black*. You also can use standard hexadecimal RGB color notation for custom colors.

Fills and strokes also may have a gradient fill, using a gradient brush. The gradient is defined by using a number of *gradient stops* across a *normalized space*. So, for example, if you want a linear gradient to move from left to right—phasing from black to white through shades of gray—you would define stops according to a normalized line. In this case, consider the beginning of the normalized line as the 0 point and the end as the 1 point. So, a gradient from left to right in a one-dimensional space has a stop at 0 and another at 1. Should you want a gradient that transitions through more than two colors—from black to red to white, for example—you would define a third stop somewhere between 0 and 1. Keep in mind that when you create a fill, however, you are working in a two-dimensional space, so (0,0) represents the upper-left corner, and (1,1) represents the lower-right corner. Thus, to fill a rectangle with a gradient brush, you would use a *LinearGradientBrush* like this:

```
<Rectangle Width="200" Height="150" >
  <Rectangle.Fill>
```

```
    <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
      <LinearGradientBrush.GradientStops>
        <GradientStop Color="Red" Offset="0" />
        <GradientStop Color="Black" Offset="1" />
      </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

XAML also supports text through the *TextBlock* element. Control over typical text properties such as content, font type, font size, wrapping, and more are available through attributes. Following is a simple example:

```
<TextBlock TextWrapping="Wrap" Width="100">
  Hello there, how are you?
</TextBlock>
```

Objects can be transformed in XAML using a number of transformations. Some of these include the following:

- ***RotationTransform*** Rotates the element through a defined number of degrees
- ***ScaleTransform*** Used to stretch or shrink an object
- ***SkewTransform*** Skews the object in a defined direction by a defined amount
- ***TranslateTransform*** Moves the object in a direction according to a defined vector
- ***MatrixTransform*** Used to create a mathematical transform that can combine all of the above

Transformations may be grouped so that you can provide a complex transformation by grouping existing ones. That is, you could move an object by translating it, change its size by scaling it, and rotate it simultaneously by grouping the individual transformations together. Here's a transformation example that rotates and scales the canvas:

```
<Canvas.RenderTransform>
   <TransformGroup>
      <RotateTransform Angle="-45" CenterX="50" CenterY="50"/>
      <ScaleTransform ScaleX="1.5" ScaleY="2" />
   </TransformGroup>
</Canvas.RenderTransform>
```

XAML supports animations through defining how their properties are changed over time using a timeline. These timelines are contained within a *storyboard*. Different types of animation include:

- ***DoubleAnimation*** Allows numeric properties, such as those used to determine location, to be animated
- ***ColorAnimation*** Allows colored properties, such as fills, to be transformed

- ***PointAnimation*** Allows points that define a two-dimensional space to be animated

As you change properties, you can do it in a linear manner, so that the property is phased between values over a timeline, or in a "key frame" manner, in which you would define a number of milestones along which the animation occurs. We'll examine all of this in a lot more detail in Chapter 5, "XAML: Transformation and Animation."

Beyond this basic XAML, you will define your full UIs using controls and layout using XAML, too. These will be explored in more detail in Chapter 7, "Silverlight Controls: Presentation and Layout," and in the rest of the chapters in Part 2 "Programming Silverlight 2."

Silverlight and the Expression Suite

Microsoft has introduced the Expression Suite of tools to provide a robust, modern set of tools for designers to express their work using artifacts that developers can include while developing using the Microsoft Visual Studio tool suite.

There are several tools in the Expression Suite:

- ***Expression Web*** This is a Web design tool that allows you to use HTML, DHTML, CSS, and other Web standard technologies to design, build, and manage Web applications.
- ***Expression Media*** This is a media asset management tool that permits you to catalog and organize these assets, including the facility to encode and change encoding between different formats.
- ***Expression Encoder*** This application is designed to allow you to manage encoding of media assets. It can also be used to bundle media with the relevant code to have a Silverlight media player for it.
- ***Expression Design*** This is an illustration and graphic design tool that you can use to build graphical elements and assets for Web and desktop application UIs.
- ***Expression Blend*** This tool is designed to let you build XAML-based UIs and applications for the desktop with WPF or for the Web with Silverlight.

When using Silverlight, you'll use some or all of these applications. In the rest of this chapter, we'll take a look at how Design, Blend, and Encoder enhance your toolkit in designing and building Silverlight applications.

Silverlight and Expression Design

Expression Design is a graphical design tool that allows you to build graphical assets for use in your applications. It's a huge and sophisticated tool, so we will just provide an overview of how it can be used for Silverlight XAML here. Expression Design allows you to blend vector-based and raster-based (bitmap) images for complete flexibility.

It supports many graphical file formats for import, such as:

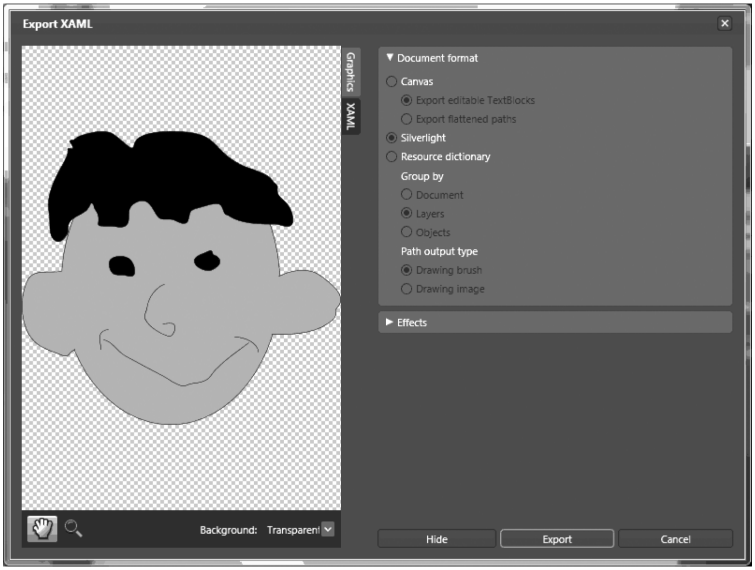
- Adobe Illustrator—PDF Compatible (*.ai)
- Adobe Photoshop (*.psd)
- Graphical Interchange Format (.gif)
- Portable Network Graphics format (.png)
- Bitmaps (.bmp, .dib, .rle)
- JPEG formats (.jpeg, .jpg, .jpe, .jfif, .exif)
- Windows Media Photos (.wdp, .hdp)
- Tagged Image File Format (.tiff, .tif)
- Icons (.ico)

It supports export of the following image types:

- XAML Silverlight Canvas
- XAML WPF Resource Dictionary
- XAML WPF Canvas
- Adobe Illustrator (.ai)
- Portable Document Format (.pdf)
- Adobe Photoshop (.psd)
- Tagged Image File Format (.tif, .tiff)
- JPEG formats (.jpeg, .jpg)
- Windows Bitmap (.bmp)
- Portable Network Graphics format (.png)
- Graphical Interchange Format (.gif)
- Windows Media Photos (.wdp)

As you can see, Expression Design supports export of graphical assets as XAML files. Later in this chapter, you'll see how to use Expression Design to design the graphical elements of a simple application, and you'll export these as XAML, which you can use in Expression Blend and Visual Studio to create an application.

Figure 1-4 shows the Export XAML dialog box in Expression Design. There are several format options, one of which is XAML Silverlight Canvas (shown selected). This option will format your drawing using the subset of XAML elements that are usable by Silverlight, allowing you to import the resulting XAML into Visual Studio or Expression Blend to build your Silverlight application.



▼ Document format

○ Canvas
- ● Export editable TextBlocks
- ○ Export flattened paths

● Silverlight

○ Resource dictionary

Group by
- ○ Document
- ● Layers
- ○ Objects

Path output type
- ● Drawing brush
- ○ Drawing image

FIGURE 1-4 Exporting XAML from Expression Design.

This will export the content as an XML document containing a *Canvas* element that contains the elements of your design. Here's a (truncated) example:

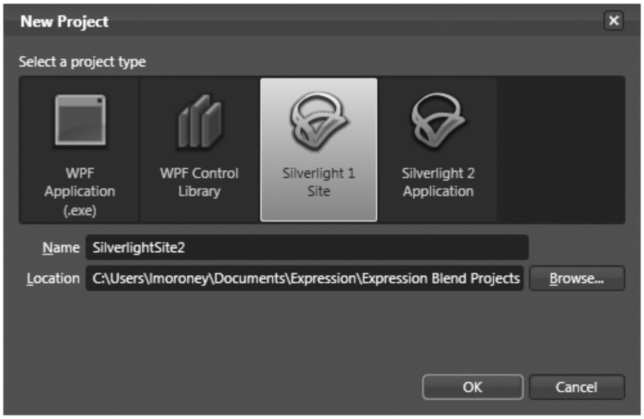
```
<?xml version="1.0" encoding="utf-8"?>
<Canvas xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" x:Name="Document">

  <Canvas x:Name="Layer_1" Width="640.219" Height="480.202" Canvas.Left="0" Canvas.Top="0">
    <Ellipse x:Name="Ellipse" Width="135" Height="161" Canvas.Left="0.546544"
      Canvas.Top="20.3998" Stretch="Fill" StrokeLineJoin="Round" Stroke="#FF000000"
      Fill="#FFFFC800"/>
    <Path x:Name="Path" Width="135.103" Height="66.444" Canvas.Left="-0.555986"
      Canvas.Top="-0.389065" Stretch="Fill" StrokeLineJoin="Round" Stroke="#FF000000"
      Fill="#FF000000" Data="..."/>
    <Path x:Name="Path_0" Width="19.4583" Height="23.9019" Canvas.Left="75.8927"
      Canvas.Top="76.1198" Stretch="Fill" StrokeLineJoin="Round" Stroke="#FF000000"
      Fill="#FF000000" Data="..."/>
    <Path x:Name="Path_1" Width="11.0735" Height="24.0564" Canvas.Left="60.473"
      Canvas.Top="106.4" Stretch="Fill" StrokeLineJoin="Round" Stroke="#FF000000"
      Fill="#FF000000" Data="..."/>
    <Path x:Name="Path_2" Width="76" Height="29.8274" Canvas.Left="31.5465"
      Canvas.Top="127.4" Stretch="Fill" StrokeThickness="7" StrokeLineJoin="Round"
      Stroke="#FF000000" Data="..."/>
    <Path x:Name="Path_3" Width="20.3803" Height="27.1204" Canvas.Left="31.2028"
      Canvas.Top="75.306" Stretch="Fill" StrokeLineJoin="Round" Stroke="#FF000000"
      Fill="#FF000000" Data="..."/>
  </Canvas>
</Canvas>
```

You can then cut and paste this XAML into Expression Blend or Visual Studio, and you will be able to use the graphical element in your application.

Silverlight and Expression Blend

Expression Blend has native support for the creation of Silverlight applications. When you launch Expression Blend and create a new project, you have two options for creating Silverlight projects, as you can see from Figure 1-5.



Name SilverlightSite2

Location C:\Users\lmoroney\Documents\Expression\Expression Blend Projects

FIGURE 1-5 Silverlight support in Expression Blend.

The two options for Silverlight projects are:

- ***Silverlight 1 Site*** This creates a Silverlight JavaScript project, giving you a folder that contains a simple Web application containing an HTML page that has the requisite scripts to embed a Silverlight object as well as a default XAML document containing a single canvas. It does not contain any of the implementation details for .NET programming, so the descriptive term 1 Site is used, even though the Silverlight control is still version 2. This will likely change in future versions of Expression Blend to Silverlight JavaScript Site. Chapter 6, "The Silverlight Browser Control," will look at programming JavaScript applications in a little more detail.
- ***Silverlight 2 Application*** This creates a Silverlight project with everything necessary to program against it using the .NET Framework. There will be more on Silverlight 2 in Chapter 3, "Using Visual Studio with Silverlight 2," and then Part 2 of this book (Chapters 7–14) will cover it in much more detail.

Exploring the Silverlight 1 Site Project

When you create a new Silverlight Script application using Blend, your project will contain a default HTML file that contains all the requisite JavaScript to instantiate the Silverlight control.

In addition, Blend also creates a basic XAML page called Page.xaml and an associated JavaScript file called Page.xaml.js. Expression Blend treats this as a "code-behind" JavaScript file in a manner that is similar to how Visual Studio treats the C# code-behind file associated with an ASPX page. Finally, Blend gives you a copy of the Silverlight.js file that is part of the Silverlight software development kit (SDK). This file manages the instantiation and downloading of the Silverlight plug-in for your users. You can see the project structure in Figure 1-6.

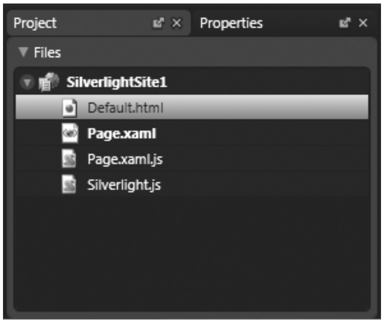


FIGURE 1-6 Project structure for a Silverlight Script application.

The Default Web Page

Listing 1-1 shows the code for the basic Web page that is created for you by Blend for Silverlight projects.

LISTING 1-1 Default.html from Silverlight Template

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!-- saved from url=(0014)about:internet -->
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>SilverlightSite1</title>

  <script type="text/javascript" src="Silverlight.js"></script>
  <script type="text/javascript" src="Page.xaml.js"></script>
  <style type="text/css">
    #silverlightControlHost {
      height: 480px;
      width: 640px;
    }
    #errorLocation {
      font-size: small;
      color: Gray;
  </style>
  <script type="text/javascript">
  function createSilverlight()
  {
    var scene = new SilverlightSite1.Page();
    Silverlight.createObjectEx({
      source: "Page.xaml",
      parentElement: document.getElementById("silverlightControlHost"),
        id: "SilverlightControl",
        properties: {
          width: "100%",
          height: "100%",
          version: "1.0"
        },
      events: {
        onLoad: Silverlight.createDelegate(scene, scene.handleLoad),
        onError: function(sender, args) {
          var errorDiv = document.getElementById("errorLocation");
          if (errorDiv != null) {
            var errorText = args.errorType + "- " + args.errorMessage;
            if (args.ErrorType == "ParserError") {
              errorText += "<br>File: " + args.xamlFile;
              errorText += ", line " + args.lineNumber;
              errorText += " character " + args.charPosition;
            }
            else if (args.ErrorType == "RuntimeError") {
              errorText += "<br>line " + args.lineNumber;
              errorText += " character " +  args.charPosition;
            }
          errorDiv.innerHTML = errorText;
          }
        }
      });
    }

  if (!window.Silverlight)
    Silverlight = {};
  Silverlight.createDelegate = function(instance, method) {
    return function() {
      return method.apply(instance, arguments);
    }
  }
</script>
</head>

<body>
  <div id="silverlightControlHost">
  <script type="text/javascript">
    createSilverlight();
  </script>
  </div>

  <div id='errorLocation'></div>
</body>
</html>
```

As you can see, it imports two JavaScript files: Silverlight.js and Page.xaml.js. You'll be looking at each of these files shortly.

The Silverlight control instantiation takes place in the *<div>* at the bottom of the page. This contains a call to the *createSilverlight* function, which is implemented at the top of the page. This creates a new Silverlight object using either the *createObjectEx* function (which, in turn, resides in Silverlight.js) or the *createObject* function. When using the *createObjectEx* function, the syntax for specifying the parameters uses the JavaScript Object Notation (JSON) syntax, as shown in this example. You can alternatively use the *createObject* function, which uses standard parameters.

The first parameter is the source XAML. This can be a reference to a static external file (which is used in this case as Page.xaml), a reference to the URL of a service that can generate XAML, or a reference to a named script block on the page that contains XAML.

The second parameter is the parent element. This is the name of the *<div>* that contains the Silverlight control. As you can see in Listing 1-1, this is called *SilverlightControlHost*.

The third parameter is the ID that you want to use for this control. If you have multiple Silverlight controls on a page, you need to have a different ID for each.

The fourth parameter is the property settings for the control properties. These can include simple properties such as width, height, and background color, as well as complex ones. More complex property settings include:

- ***inplaceInstallPrompt*** Determines the install type for Silverlight. If this is set to *true*, the user implicitly accepts the license and directly downloads and installs the plug-in. If it is set to *false*, the user is directed to *http://www.silverlight.net* and, from that site, can accept the license and download the plug-in.
- ***isWindowless*** If set to *true*, the control is considered *windowless*, meaning that you can overlay non-Silverlight content on top of it.
- ***framerate*** Determines the maximum frame rate for animations.
- ***version*** Determines the minimum Silverlight version your application will accept. As you can see in Listing 1-1, the version is listed as 1.0—this isn't a bug, but simply an instruction that this application is backward compatible and should work on 1.0. If this was instead 2.0, and Silverlight 2 was not installed, then the user would be taken to the install experience for Silverlight 2.

The fifth parameter is used to map events to event handlers. The events are implemented in a JavaScript class called *scene,* which was declared at the top of the function:

```
var scene = new SilverlightSite1.Page();
```

The *createSilverlight* function declares that the *onLoad* event should be handled by a member function of the *scene* class called *scene.handleLoad*. It does this by creating a delegate using this syntax:

```
onLoad: Silverlight.createDelegate(scene, scene.handleLoad)
```

This class is implemented in the JavaScript code-behind for Page.xaml called Page.xaml.js. You can see this in Listing 1-2.

LISTING 1-2 JavaScript Code-Behind Page.xaml

```
if (!window.SilverlightSite1)
  SilverlightSite1 = {};

SilverlightSite1.Page = function()
{
}

SilverlightSite1.Page.prototype =
{
  handleLoad: function(control, userContext, rootElement)
  {
    this.control = control;

    // Sample event hookup:
    rootElement.addEventListener("MouseLeftButtonDown",
        Silverlight.createDelegate(this, this.handleMouseDown));
  },

  // Sample event handler
  handleMouseDown: function(sender, eventArgs)
  {
    // The following line of code shows how to find an element by name
this.control.content.findName("Storyboard1").Begin();
  }
}
```

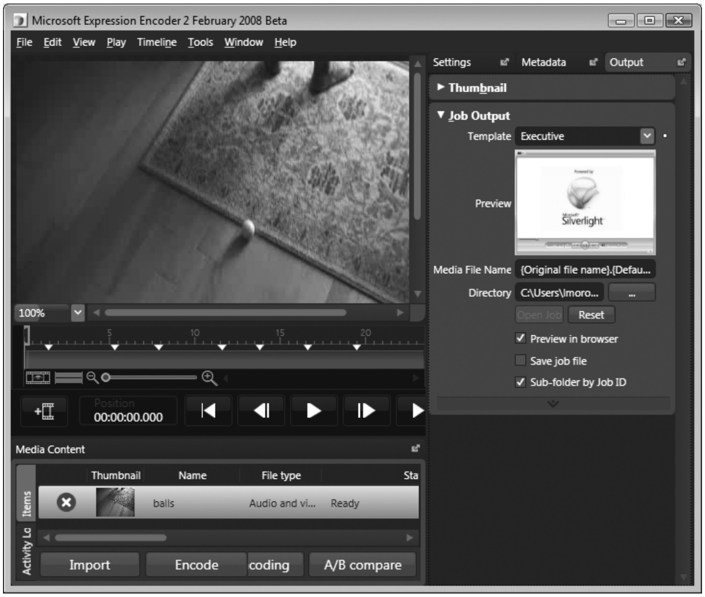
Here you can see JavaScript code to create a class called *SilverlightSite1.Page*. It contains two member functions, *handleLoad* and *handleMouseDown*.

The function *handleLoad* adds another event listener for the *MouseLeftButtonDown* event by creating a delegate associating this event and the *handleMouseDown* function, which is also defined within this JavaScript.

Thus, the template application creates a default HTML file that contains an instance of Silverlight with a single canvas that fires an event when it loads. The load event wires up the mouse down event, demonstrating that event declaration, delegation, and handling are available at both design time and run time.

Silverlight and Expression Encoder

Expression Encoder is an application that allows you to encode, enhance, and publish your video content using Silverlight. It comes with a UI that is consistent with the rest of the Expression suite or with a command-line interface that can be used for batch work. You can see Expression Encoder in Figure 1-7.



Template: Executive
Media File Name: {Original file name}.{Defau...
Directory: C:\Users\lmoro...
☑ Preview in browser
☐ Save job file
☑ Sub-folder by Job ID

balls — Audio and vi... — Ready

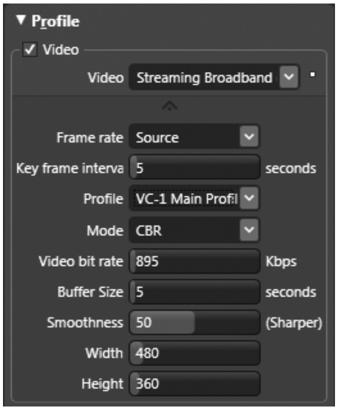
FIGURE 1-7 Expression Encoder.

Expression Encoder allows you to import video from any format for which a DirectShow filter is available and installed in your system. It will then re-encode the video into a VC-1–capable WMV using one of a number of preset profiles optimized for the delivery client. These include settings for devices, as well as for streaming or on-demand content delivered over the Internet.

You aren't limited to what the preset profiles give you—you can override any of the video and audio encoding settings. Figure 1-8 (on page 20) shows an example of how a video encoding may be tweaked.

Media Encoder includes a number of preset media player applications for Silverlight. These will "wrap up" your video with a Silverlight JavaScript-based application that can be used on any Web server to provide a complete Silverlight-based viewing experience.

In addition to encoding, metadata can be added to your video. A classic metadata experience is when tags are encoded into the video and the application then reacts to these tags. Inserting tags with Expression Encoder is very simple. Simply drag the playhead to the desired point, select Add Marker, and enter the appropriate information for the marker.



▼ Profile

☑ Video

Video: Streaming Broadband

Frame rate: Source

Key frame interva: 5 seconds

Profile: VC-1 Main Profil

Mode: CBR

Video bit rate: 895 Kbps

Buffer Size: 5 seconds

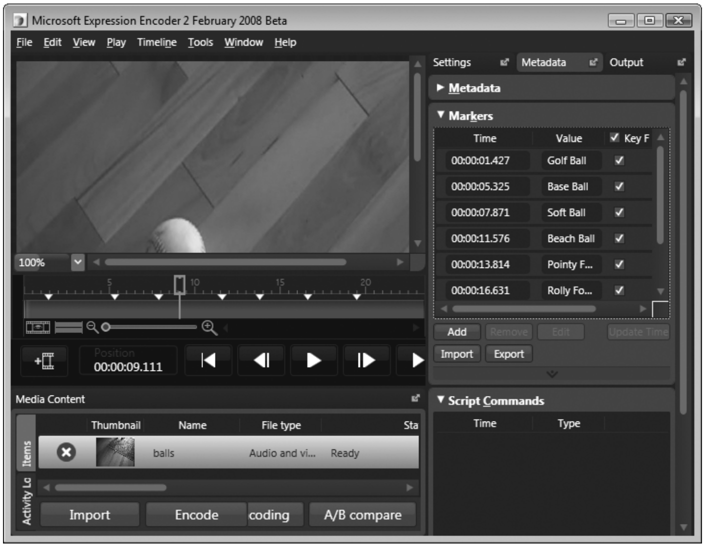
Smoothness: 50 (Sharper)

Width: 480

Height: 360

FIGURE 1-8 Configuring a video encoding profile.

You can see this in Figure 1-9 on the right side of the screen, where the marker time and type of ball that is shown on the screen at that time has been configured.



Time	Value	☑ Key F
00:00:01.427	Golf Ball	☑
00:00:05.325	Base Ball	☑
00:00:07.871	Soft Ball	☑
00:00:11.576	Beach Ball	☑
00:00:13.814	Pointy F...	☑
00:00:16.631	Rolly Fo...	☑

100%

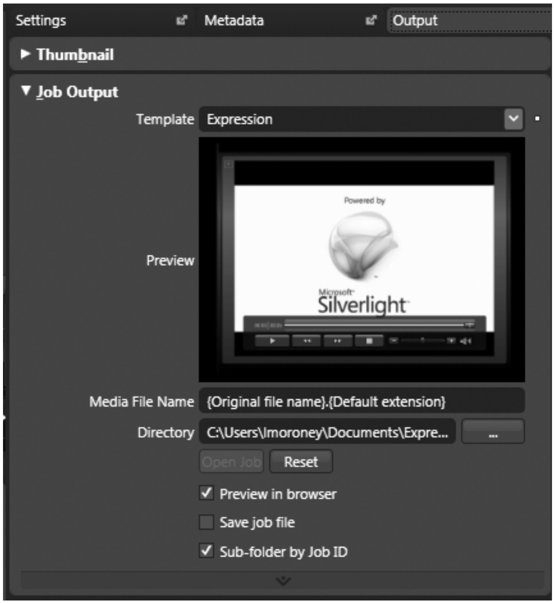
Position 00:00:09.111

Thumbnail	Name	File type	Sta
	balls	Audio and vi...	Ready

FIGURE 1-9 Adding Markers to a stream.

The Output tab allows you to select the template player that you want to use.

Figure 1-10 shows where the template that matches the Expression product line has been selected. To create a video player with this template, simply import a video, and press the Encode Button with this template selected.



Template: Expression

Media File Name: {Original file name}.{Default extension}

Directory: C:\Users\lmoroney\Documents\Expre...

☑ Preview in browser

☐ Save job file

☑ Sub-folder by Job ID

FIGURE 1-10 Using Encoder to build a Silverlight media player.

After you've done this, you'll get a full-featured media player in Silverlight for your video content. You can see an example of a Silverlight media player in Figure 1-11.

This section just scratches the surface of what is possible with Expression Encoder and how it can be used with Silverlight. For more details, please refer to *http://www.microsoft.com/expression*.

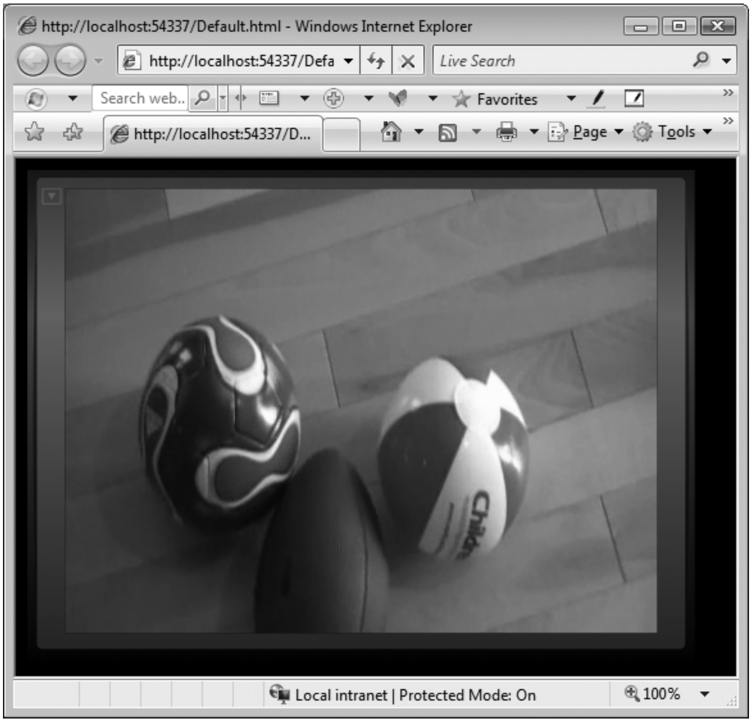

http://localhost:54337/Default.html - Windows Internet Explorer
http://localhost:54337/Defa
Live Search
Search web..
Favorites
http://localhost:54337/D...
Page
Tools
Local intranet | Protected Mode: On
100%

FIGURE 1-11 Media player generated by Expression Encoder.

Summary

In this chapter, you were introduced to Silverlight 2 and learned how it fits into the overall Web and UX landscape. You discovered how technology from Microsoft is applied to current UX scenarios, and you were introduced to an overview of the Silverlight architecture, including XAML and how it is used to implement rich UIs.

Additionally, you saw how the Microsoft Expression Suite is designed to complement traditional development tools such as Visual Studio for creating Silverlight applications. You specifically learned how Expression Design is used to build graphical assets and how Expression Blend is used to link these together into an interactive application as well as using Expression Encoder to manage your video assets.

Now it's time to go deeper. In the next few chapters, you'll learn more about the Silverlight API, starting with a more detailed examination of Expression Blend and how it is used by Silverlight in the next chapter.