Microsoft®

INTRODUCING MICROSOFT® SILVERLIGHT™ 1.0

Laurence Moroney

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2007938162

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 QWT 2 1 0 9 8 7

Distributed in Canada by H.B. Fenn and Company Ltd.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at www.microsoft.com/mspress. Send comments to mspinput@microsoft.com.

Acquisitions Editor: Ben Ryan
Developmental Editor: Devon Musgrave
Project Editor: Victoria Thulman
Editorial Production: Custom Editorial Productions, Inc.
Technical Reviewer: Kenn Scribner

Body Part No. X14-14867

Contents at a Glance

Table of Contents

Introduction

Why Silverlight?

As the Web grows and evolves, so do the expectations of the Web user. When the first Web browser was developed, it was created to provide a relatively simple way to allow hyperlinking between documents. Then these early browsers were coupled with the cross-machine protocols encompassing the Internet, and suddenly documents stored on computer servers anywhere in the world could be hyperlinked to each other.

Over time, the people who were using the Internet changed—the user base expanded from a small group of people associated with universities and computational research to encompass the general population. And what had been an acceptable user interface for experts in the field was greatly lacking for commercial applications. People want high-quality user interfaces that are simple to use—and as more types of information, including many kinds of media files, are available on the Internet, it becomes more difficult to satisfy users' expectations about how easy it should be to access the information they want.

The need to supply users with sophisticated methods of accessing Internet resources that were easy to use led to advanced application technologies. One type of technology, for example, created "plug-in" browser tools that allowed the browser to use some of the user's local computational horsepower. ActiveX controls, Java Applets, and Flash applications are examples of plug-in technology. Asynchronous JavaScript and XML (AJAX) is another tool that has been introduced to develop new and exciting user interfaces that benefit from immediate partial updates. Using AJAX, the browser's screen area doesn't flash or lock up, since the need for full-page refreshes is reduced.

Although AJAX provides technology to enable developers to build Web sites that contain more complex content and are more dynamic than HTML alone could provide, AJAX does have its limitations. For example, it allows asynchronous communication with the server, which means that applications can update themselves using background threads, eliminating the screen flicker so often seen with complex Web user interfaces. But AJAX is strictly a browser-to-server communications mechanism. It lacks graphics, animation, video, and other capabilities that are necessary to provide for truly multimedia user interfaces.

Microsoft has built a Web user experience (UX) strategy to address these limitations by identifying three levels of desired user experience—"good," "great," and "ultimate," which are mapped to development and run-time technologies. These are combined in this book with a term you may find that I use a lot—"rich" or "richness." When I say "rich," I'm trying to describe a concept that's hard to put into words. It's the feeling you get when you use a traditional Web application, with the limitations built into the browser and HTML, versus a desktop application that has the entire operating system to call on for services and capability. The

Web applications of today just don't have the same feeling and capability as desktop applications, and the user generally realizes that they are limited by the technology. With Silverlight (and AJAX), the goal is to create Web applications that are much more like desktop applications, and ultimately, to create applications that are indistinguishable from desktop applications.

The lowest level of user experience, the "good" level, can be achieved with the browser enhanced by AJAX. This level identifies the baseline UX expectation moving forward from today—the type of asynchronous, dynamic, browser application empowered by AJAX.

The top or "ultimate" level is the rich client desktop, running Windows Vista and using the Windows Presentation Foundation (WPF) and the .NET Framework. These offer a run time that allows developers to create extremely rich applications that are easily deployed and maintained. Broadcast quality graphics, video, and animation are available at this level, as well as application services such as file-based persistence and integration with other desktop applications. In addition, WPF separates design and development technologies, so that user interfaces are designed and expressed in a new language called XML Application Markup Language (XAML). Design tools such as the Microsoft Expression series were aimed at designers who are now able to produce their work as XAML documents. Developers then use the resulting XAML to bring the designers' dreams to reality more easily by activating the XAML with code.

I mentioned that there are three levels in the UX strategy, because as AJAX and .NET/WPF evolved, it was obvious that there was room in the middle for a new technology that effectively takes the best of both worlds—the global scalability of the Internet application coupled with the richness of the desktop application. This level was named the "great" experience and represents the browser enhanced by AJAX with a new technology: Silverlight.

Silverlight is a plug-in for the browser that renders XAML and exposes a JavaScript programming interface. Thus, it allows designers and developers to collaborate when building Internet applications that provide the richness of desktop applications.

In this book, you'll be looking at Silverlight 1.0 and how to use it to enhance Web user experience. You'll also have a chance to take a look at Silverlight 1.1 and learn how this new version will allow you to further enhance your Web experience through the programming power of the .NET Framework.

Silverlight can change the way you think about building applications for the Web. Instead of Web sites, you will build Web *experiences*. At the heart of a great experience is great design, and with Silverlight, designers and developers can come together like never before, through XAML and the Microsoft Expression line of tools.

In this book, my goal is to help you understand the technologies that work together to develop and deploy a complete Silverlight Web application, from writing basic code that uses Silverlight to using advanced tools to create and deliver Silverlight content. When you have finished reading this book and have worked the examples, you should be ready to use what you've learned to enhance the Web applications you're developing *right now*. Imagine what you'll be able to do *tomorrow*!

Who This Book Is For

This book is written for developers who are already working every day to bring new and better Web applications to Internet users, and who are interested in adding this cutting-edge Microsoft technology to their store of knowledge—to find out how it can be applied as a tool to bring users more interesting, more capable, and more effective user interfaces. Development managers may also find the easy-to-read style useful for understanding how Silverlight fits into the bigger Microsoft Web technology picture, and with luck, will provide them with the technological background they need, so that when their developers come to them to talk about Silverlight—with excited looks on their faces—the manager will understand what the excitement is about!

What This Book Is About

In Chapter 1, "Introducing Silverlight," you'll get an introduction to Silverlight and take a quick tour of the features available in version 1.0. You'll also build some simple applications.

Chapter 2, "Silverlight and XAML," will introduce you to XAML and show you how you can use it to define your Web applications.

XAML and Silverlight offer APIs for managing timeline-based animation, which you will look into in some detail in Chapter 3, "XAML: Transformation and Animation." Expectations for the great user experience will involve using audiovisual media, and Chapter 4, "Silverlight and Media," offers an introduction to the media features available in Silverlight.

JavaScript is the programming heart of Silverlight 1.0, and you'll look into the JavaScript programming APIs in Chapter 5, "Programming Silverlight with JavaScript."

Silverlight supports Ink computing, which allows for new form factors when interfacing with the Web. You'll look into how Ink works with Silverlight in Chapter 6, "Using Silverlight with Ink."

Chapter 7, "Silverlight Server Programming," provides a look at how server applications can interact with and support Silverlight applications, and how Silverlight's open nature makes it ideal for cross-platform applications. You will see how to build server applications that deliver Silverlight content using Java, PHP, and ASP.NET.

Finally, in Chapter 8, “Silverlight Futures,” you’ll take a look into the future to find out how Silverlight is changing and growing. You’ll see how the new “mini” .NET CLR will allow you to code high-performing applications for Silverlight, and how Silverlight can be extended with custom controls.

By the time you’ve finished reading this book, you will have a firm grip on what you need to build Silverlight applications. This book is designed to get you up and running quickly. I hope you’ll have as much fun reading it as I did writing it!

Developing for Silverlight: System Requirements

Downloads for all the tools that you’ll need to build Silverlight 1.0 and Silverlight 1.1. applications are listed here and are generally available at *http://silverlight.net/GetStarted/* (if they’re not available at this site, you’ll find links there to sites where they are available). The Silverlight Runtime for Windows is supported on Windows XP, Windows 2003, and Windows Vista using either the FireFox browser (1.5 and later) or Internet Explorer (6 and later). The Silverlight Runtime for Mac is supported on Mac OS 10.4.8 and later using either the FireFox browser (1.5 and later) or Safari.

To develop Silverlight applications as used in this book, you will need the following (again, available at *http://silverlight.net/GetStarted/*):

- Microsoft Visual Studio 2008
- Microsoft Expression Design
- Microsoft Expression Blend
- Microsoft Silverlight 1.0 Software Development Kit

Some of the book’s samples will need the following tools:

- Microsoft ASP.NET Futures
- Microsoft Silverlight Tools for Visual Studio
- Microsoft Silverlight 1.1 Software Development Kit

The Companion Web Site

This book features a companion Web site that makes available to you all the code used in the book. This code is organized by chapter, and you can download it from the companion site at this address:

http://www.microsoft.com/mspress/companion/9780735625396

Support for This Book

Microsoft Press provides support for books and companion content at the following Web site:

http://www.microsoft.com/learning/support/books/

Questions and Comments

If you have comments, questions, or ideas regarding the book or the companion content, or questions that are not answered by visiting the sites just listed, please send them to Microsoft Press via e-mail to

mspinput@microsoft.com

Or via postal mail to

Microsoft Press

Attn: *Introducing Microsoft Silverlight 1.0* Editor

One Microsoft Way

Redmond, WA 98052-6399

Please note that Microsoft software product support is not offered through the above addresses.

Chapter 1
Introducing Silverlight

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 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 without compilation and will be based on XML, making it easy to template and to automatically generate. The format is XAML—XML 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 and 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 way of actual Web technologies including AJAX, JavaScript and DHTML, and Silverlight 1.1 adds dynamic and .NET language support. This book will concentrate on version 1.0, but Chapter 8, "Silverlight Futures," does provide a peek into the future of Silverlight, including how to program it using C#, Ruby, and Python.

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 browser (no plug-ins)

- Enhanced browser (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 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.

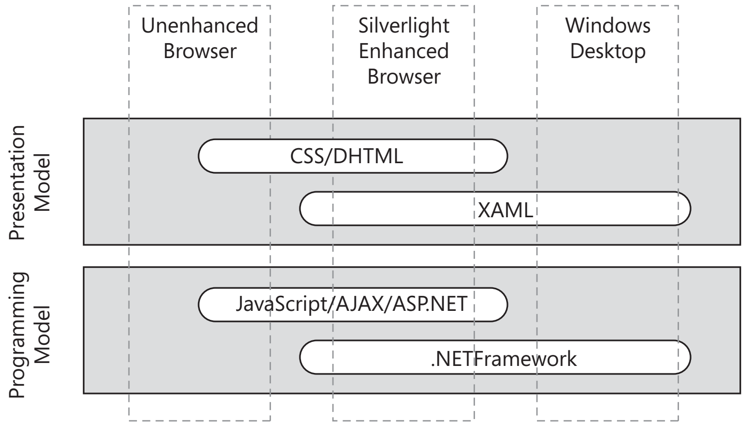


Figure 1-1 Programming and presentation models for the Web.

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 user interfaces (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 XML, and 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 graft 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 exposes its internal Document Object Model (DOM) and event programming model to the browser in a way that is scriptable via JavaScript.

The architecture that supports this is shown in Figure 1-2. 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, media playback to be controlled or animations to be triggered.

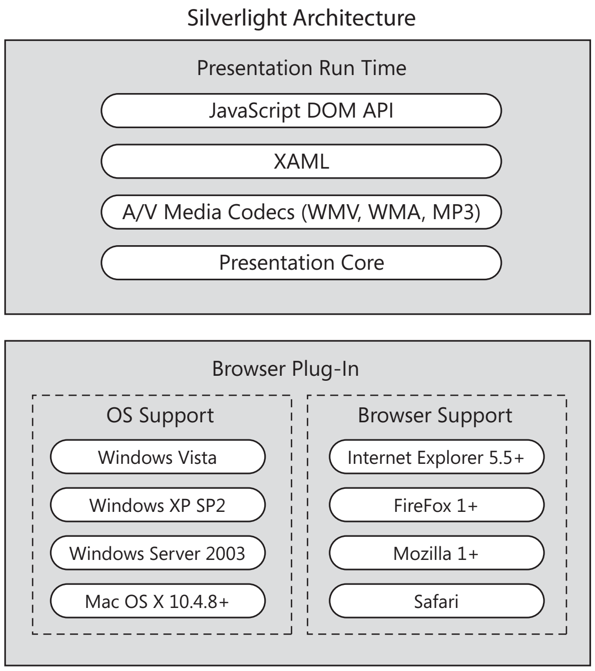


Figure 1-2 Silverlight architecture.

Additionally, the presentation run time 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 run time 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.

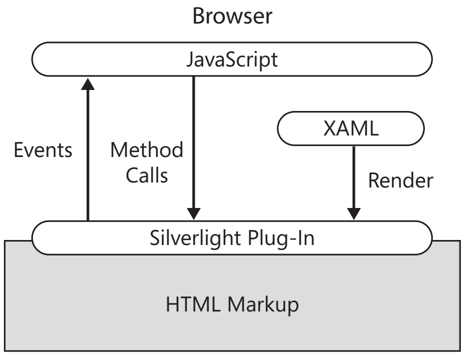


Figure 1-3 Application architecture with Silverlight.

As the application runs within the browser, it is typically made up of HTML markup. 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 functions written in JavaScript. In turn, program code written in JavaScript 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.

Your First Silverlight Application

What introduction would be complete without a "Hello World" application? In this section, you'll have the chance to look at all the pieces that make up a Silverlight application and how they work together. Although the application you build will be very simple, the principles of building a much more complex application are the same–and you'll be well on your way to understanding Silverlight development! You'll need no special tools–everything you do can be done with a simple text editor. You'll revisit these files again in Chapter 5, "Programming Silverlight with JavaScript."

Step 1: Silverlight.js

The first thing you'll need is the Silverlight.js file. This file contains everything you need to create a Silverlight component on your page. Silverlight is a browser plug-in that renders XAML and exposes a JavaScript programming interface. Browser plug-ins are implemented using special HTML tags called *object* and *embed*. Different browsers handle them differently, so instead of having to adjust them for a particular type of browser, it's a lot easier just to use the Silverlight.js file, which deals with the different browser implementations for you. It is available in the Silverlight Software Development Kit (SDK), which by default installs to \PROGRAM FILES\Microsoft Silverlight 1.0 SDK. You'll find Silverlight.js in the Resources directory. To use the file, simply include it with your Web project and provide a link on any page that will host the Silverlight control (you'll see this in Step 5 later in this section).

Step 2: XAML

A Silverlight user interface (UI) is defined using XAML—XML Application Markup Language. Some great resources to get you started with XAML can be found in the Silverlight Quick-Starts, task-based examples that provide tutorials to help you learn the features of Silverlight. The QuickStart tutorials are available at *http://www.silverlight.net/quickstarts*.

Our simple first application will use a XAML *Canvas* that contains a *TextBlock* control which, as its name suggests, renders text:

```
<Canvas
    xmlns="http://schemas.microsoft.com/client/2007"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="640" Height="480"
    Background="White"
    x:Name="Page">
    <TextBlock Width="195" Height="42" Canvas.Left="28" Canvas.Top="35"
          Text="Hello World!" TextWrapping="Wrap" x:Name="txt"/>
</Canvas>
```

This simple piece of XAML contains two components. The first component is the root *Canvas* element, which is present in every Silverlight XAML and defines the overall drawing surface. As you can see, we are using a 640 × 480 white *Canvas*. The second component is the *TextBlock* element I mentioned earlier. It renders the text "Hello World!" on the *Canvas*. This new XAML document should be saved and named. For this example, use the file name Page.xaml.

Keep in mind that XAML is just XML, so all of the conventions of XML apply. You can see that the *TextBlock* is a child node of the *Canvas*, and that XML attributes are used to define the properties of the objects. This allows for some cool scenarios, such as generating UI on demand from server applications using ASP.NET, Personal Home Page (PHP), or Java. We'll look at some of these possibilities later in this book.

Step 3: CreateSilverlight.js

It's good practice to host the code for creating the Silverlight component on your page in a separate JavaScript file. Although not essential, it is a useful step that promotes clean separation of code.

By convention, you would name this file CreateSilverlight.js, but of course you can name it anything you like as long as you reference it correctly when you assemble your HTML. Following is an example function that instantiates the Silverlight control in the browser:

```
function createSilverlight()
{
    Silverlight.createObjectEx({
        source: "Page.xaml",
        parentElement: document.getElementById("SilverlightControlHost"),
        id: "SilverlightControl",
        properties: {
            width: "100%",
            height: "100%",
            version: "1.0"
        },
        events: {
            onLoad: handleLoad
        }
    });
}
```

This function calls the *createObjectEx* function, which is implemented in Silverlight.js, an essential file that you added to your site in Step 1 of this example. You'll notice that an event handler, *handleLoad*, has been added to handle the *Load* event. You'll see how this is implemented in Step 4.

Step 4: Your Application Logic

This simple application allows you to click on the text block and cause the text to change from "Hello World!" to "You clicked me!" The code for this application is shown here:

```
var SilverlightControl;
var theTextBlock;
function handleLoad(control, userContext, rootElement)
{
    SilverlightControl = control;
    theTextBlock = SilverlightControl.content.findName("txt");
    theTextBlock.addEventListener("MouseLeftButtonDown", "txtClicked");
}
function txtClicked(sender, args)
{
    theTextBlock.Text = "You clicked me!";
}
```

The *handleLoad* method was defined as an event handler in the *createSilverlight* function. When Silverlight renders the control, it calls this method, passing it a reference to the control, the contents of the *userContext* variable (which can be set in the createSilverlight), and a reference to the root canvas element. You'll see all of this again in more detail in Chapter 5.

The *handleLoad* method locates the text block (named *txt*) and adds an event listener to its listener collection. The event that it is listening for is *MouseLeftButtonDown*, and when this event fires, the *txtClicked* function is invoked and the text is changed accordingly. This code should be saved to a new file named code.js and included in your project.

When you implement an event handler, as I have done with *handleLoad*, your function should accept parameters for *sender* (the originator of the event) and *args* (arguments associated with the event).

Step 5: Your HTML Page

Now it's time to put it all together with an HTML page that references each of the JavaScript files and embeds the Silverlight control. Following is the full HTML markup:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3c.org/TR/1999/REC-html401-19991224/loose.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>ZeroHero</title>
        <script type="text/javascript" src="Silverlight.js"></script>
        <script type="text/javascript" src="CreateSilverlight.js">
    </script>
        <script type="text/javascript" src="code.js">
    </script>
        <style type="text/css">
            .silverlightHost {
                height: 480px;
                width: 640px;
            }
        </style>
    </head>

    <body>
        <div id="SilverlightControlHost" class="silverlightHost">
            <script type="text/javascript">
                createSilverlight();
            </script>
        </div>
    </body>
    </html>
```

Upload all these files to your Web server and you're done.

This might have seemed to be a lot of work just to get a "Hello World" application working, but it also introduced you to the general principles involved with developing a Silverlight 1.0 application. You saw how to use Silverlight.js and CreateSilverlight.js, write XAML, load XAML into Silverlight, hook up events, and create run-time event handlers. The remainder of this book will examine those topics in more detail.

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 user interfaces, graphical assets, animations, media, controls, and more. It was introduced by Microsoft for the Windows Presentation Foundation (formerly Avalon), which is a desktop-oriented technology and part of the .NET Framework 3.0. It's designed, as discussed earlier, to bridge the gap between designers and developers when creating applications.

The XAML used in Silverlight 1.0 differs from that in the Windows Presentation Foundation in that it is a *subset* that is focused on Web-oriented features. So, if you're familiar with XAML from the Windows Presentation Foundation, you'll notice some missing tags and functionality, such as the *<Window>* element, data binding, and the rich control model.

XAML uses XML to define the UI using XML elements. At the root of every Silverlight XAML document is a *Canvas* element that defines the space on which your UI will be drawn. This root *Canvas* element 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 JavaScript 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 user interface 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 *Canvas* also contains a *Rectangle* and a *Canvas*, and this final canvas contains two more rectangles. This hierarchical structure allows for controls to be logically grouped together 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 2, "Silverlight and XAML," 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 or *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 right to left—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-hand corner, and (1,1) represents the lower right-hand 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 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 Visual Studio tool suite.

There are four main 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 Design** This is an illustration and graphic design tool that you can use to build graphical elements and assets for Web and desktop application user interfaces.
- **Expression Blend** This tool is designed to let you build XAML-based user interfaces and applications for the desktop with WPF or for the Web with Silverlight.

In this chapter, we will take a look at Expression Design and Expression Blend and how they can used to build XAML for use in Silverlight.

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 this is just an overview of how it can be used for Silverlight XAML. 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:

Windows Metafile and enhanced Metafile (.wmf, .emf)

- Photoshop (.psd)

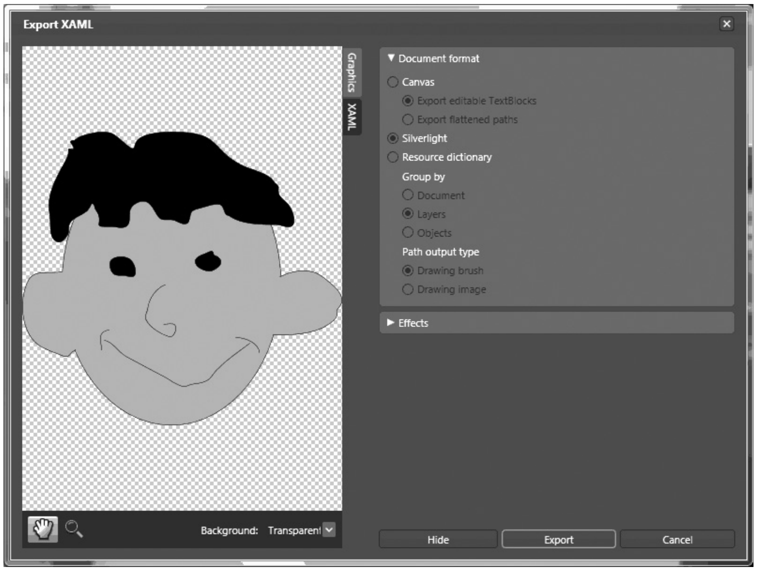
- Graphical Interchange Format (.gif)
- PNG format (.png)
- Bitmaps (.bmp, .dib, .rle)
- JPEG formats (.jpeg, .jpg, .jpe, .jfif, .exif)
- Windows Media Photos (.wdp, .hdp)
- Tagged Image File Format (.tiff, .tif)
- Icons (.ico)

It supports export of the following image types:

- XAML
- Encapsulated Postscript (.eps)
- Adobe Illustrator (.ai)
- Portable Document Format (.pdf)
- Adobe Photoshop (.psd)
- Tagged Image File Format (.tif, .tiff)
- JPEG formats (.jpeg, .jpg)
- Windows Bitmap (.bmp)
- PNG 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 media player, 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 Silverlight (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

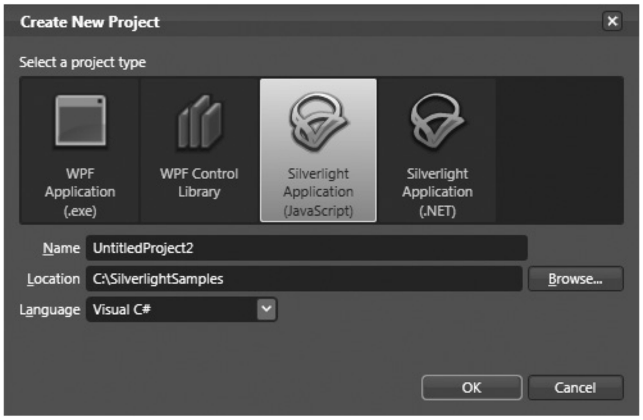
► Effects

Background: Transparent

Figure 1-4 Exporting XAML from Expression Design.

Silverlight and Expression Blend

Expression Blend 2.0 has native support for the creation of Silverlight 1.0 and Silverlight 1.1 applications. When you launch Expression Blend and create a new project, you have three options for creating Silverlight projects, as you can see from Figure 1-5.



Select a project type
- WPF Application (.exe)
- WPF Control Library
- [✓ Silverlight Application (JavaScript)]
- Silverlight Application (.NET)

Name: UntitledProject2
Location: C:\SilverlightSamples
Language: Visual C#

Figure 1-5 Silverlight support in Expression Blend.

The two options for Silverlight projects are:

- **Silverlight Application (JavaScript)** This creates a Silverlight 1.0 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.
- **Silverlight Application (.NET) Application** This creates a project for Silverlight 1.1. At present this is in prerelease, and this project template will give you the opportunity to try it out. There will be more on Silverlight 1.1 in Chapter 8.

This book will primarily cover Silverlight 1.0 JavaScript applications. Chapter 8, looking at the future of Silverlight, will discuss how the ASP.NET controls and .NET applications will be constructed using Silverlight.

Chapter 5 examines Silverlight 1.0 programming in JavaScript in more detail. We'll take a quick tour through a basic Silverlight 1.0 template application in the rest of this section, but more detail regarding this code and how to build your own is available in Chapter 5.

When you create a new Silverlight 1.0 Script application, your project will contain a default HTML file with an associated JavaScript file that is named Default.html.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. Blend also creates a Scene.xaml file and a Scene.xaml.js file. Finally it gives you a copy of the Silverlight.js file that is part of the Silverlight 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.

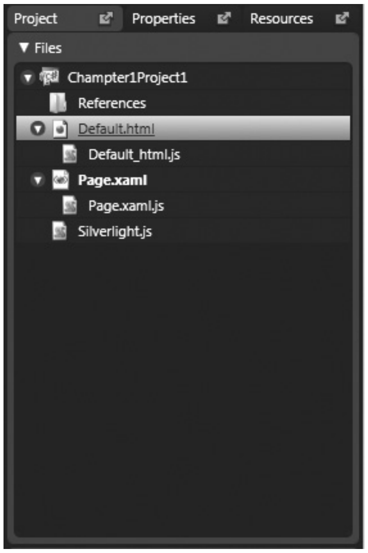


Figure 1-6 Project structure for Silverlight Script application.

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 HTML 4.01 Transitional//EN"
"http://www.w3c.org/TR/1999/REC-html401-19991224/loose.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Chapter1Project1</title>

   <script type="text/javascript" src="Silverlight.js"></script>
   <script type="text/javascript" src="Default.html.js"></script>
   <script type="text/javascript" src="Page.xaml.js"></script>
   <style type="text/css">
     .silverlightHost {
      height: 480px;
         width: 640px;
      }
   </style>
</head>

<body>
   <div id="SilverlightControlHost" class="silverlightHost">
      <script type="text/javascript">
         createSilverlight();
      </script>
   </div>
</body>
</html>
```

As you can see, it imports three JavaScript files: Silverlight.js, Default.html.js, and Page.xaml.js. You'll be looking at each of these files later in this chapter.

The Silverlight control instantiation takes place in the <div> at the bottom of the page. This contains a call to a script called *createSilverlight*, which is implemented in Default.html.js. This is the typical programming pattern used for Silverlight applications—the instantiation of the control is contained in an external JavaScript script. This keeps your page code much cleaner.

Listing 1-2 contains the JavaScript code from Default.html.js.

Listing 1-2 Default.html.js Code-Behind

```
function createSilverlight()
{
   var scene = new Chapter1Project1.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)
      }
   });
}


if (!window.Silverlight)
   window.Silverlight = {};

Silverlight.createDelegate = function(instance, method) {
   return function() {
      return method.apply(instance, arguments);
   }
}
```

This code contains the createSilverlight function that you saw referred to in Default.html.. This can create a new Silverlight object using either the createObject or the createObjectEx function. When using the latter function, the syntax for specifying the parameters uses the JavaScript Object Notation (JSON) syntax as shown in this example.

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 *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 that for animations.
- ***version*** Determines the minimum Silverlight version your application will accept.

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 Chapter1Project1.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 1-3.

Listing 1-3 JavaScript Code-Behind Page.xaml

```
if (!window.Chapter1Project1)
   window.Chapter1Project1 = {};

Chapter1Project1.Page = function()
{
}

Chapter1Project1.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 and
      // calls a method.
      // this.control.content.findName("Timeline1").Begin();
   }
}
```

Here you can see JavaScript code to create a class called *Chapter1Project1.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 script.

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.

In the next section, you'll build out a simple media player application that puts this theory into action. More detail on how to use JavaScript to program Silverlight applications is available in Chapter 5.

Building a Silverlight Media Player

It's always best to put the theory into practice with some hands-on experience, so for the rest of this chapter, you'll see how to use what you've learned to create a full Silverlight application.

You'll use all of the tools that we've discussed so far. It's not *necessary* to do this—you could do everything with Visual Studio or Expression Blend—but it's useful to demonstrate the tools continuum and how the tools are optimized for the workflow of building an application for the User Experience.

You'll build out a simple video player, using graphical assets designed with Expression Design. As Expression Design is capable of using raster and vector graphics, you'll create the look and feel of the video player in Expression Design, and then Export this as XAML. You'll then take the XAML and load it into Blend, where you will add the video control and wire it up to a video file. You'll also implement the code that handles the play, stop, and pause controls.

Designing the Assets in Expression Design

To get started, let's create a new graphic in Expression Design by selecting New from the File menu. This opens a dialog box that you can use to create a new image and specify its dimensions and resolution. Figure 1-7 shows an example of a 500 × 500 pixel design to which a bitmap of the Silverlight logo has been added.

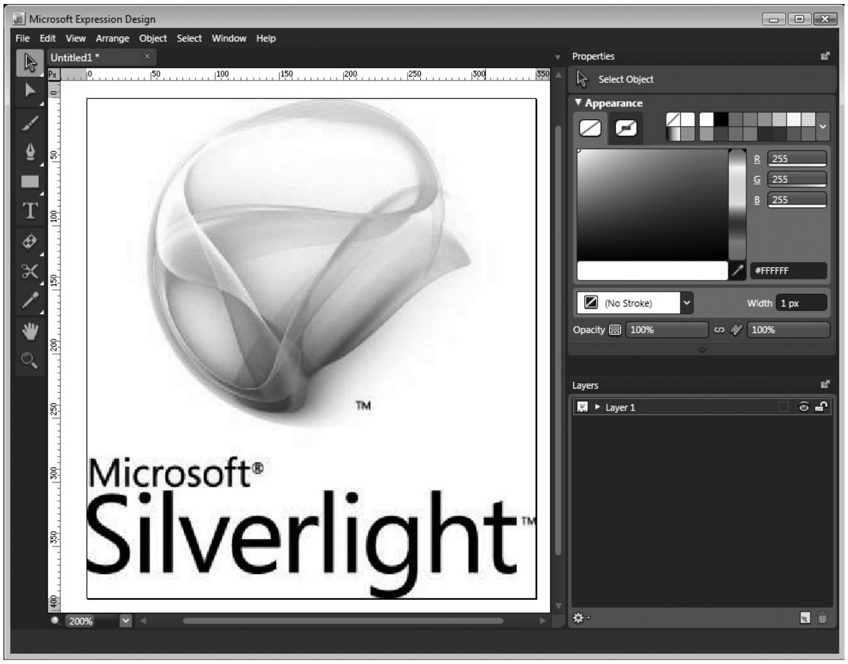


Figure 1-7 Starting to design the video player in Expression Design.

The next thing to do is to add some vector graphic assets to represent the controls for the video playback. On the left side of the screen, you'll see the controls toolbar, and on this there is a list of available shapes. Hold the mouse down on the rectangular shape, and you'll get a pop-up palette of available shapes, as shown in Figure 1-8.

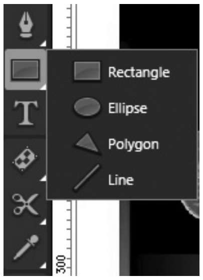


Figure 1-8 Shapes tools palette in Expression Design.

From the palette, choose Ellipse and draw a circle on the design surface. For this example, draw a small circle under the Silverlight image on the bitmap. You'll see the red outline of a circle drawn on the surface of the graphic. See Figure 1-9 for an example.

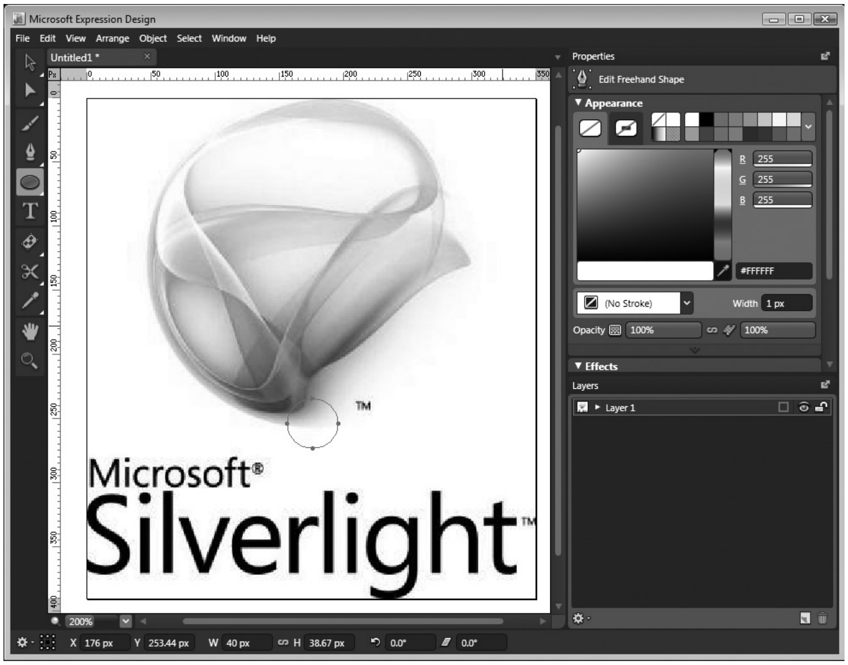


Figure 1-9 Adding a vector ellipse.

Move over to the Appearance dialog box and select the stroke and fill that you want to use. Figure 1-10 shows the Appearance dialog box with a gradient fill and a white stroke selected for the circle.

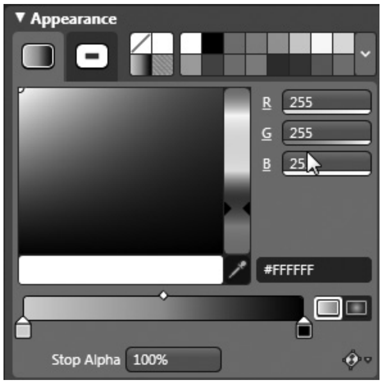


Figure 1-10 Editing the Appearance properties.

If you select similar options, you will see a gradient-filled circle surrounded by a white border. Use the Copy and Paste commands to make two more instances of the circle and put them on either side of the first. Your screen should now look something like Figure 1-11.

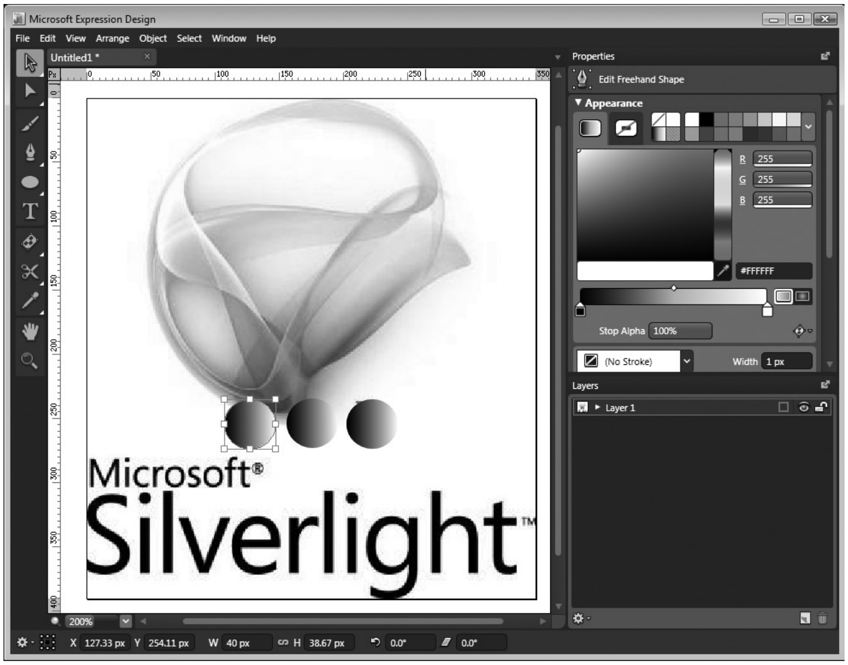


Figure 1-11 Three copies of the filled ellipse.

Now use the polygon tool to draw a triangle within the center circle and then use the rectangle tool to draw a square within the circle on the left and two tall rectangles within the circle on the right to create the traditional Play, Stop, and Pause icons, respectively. Figure 1-12 shows the completed icons.

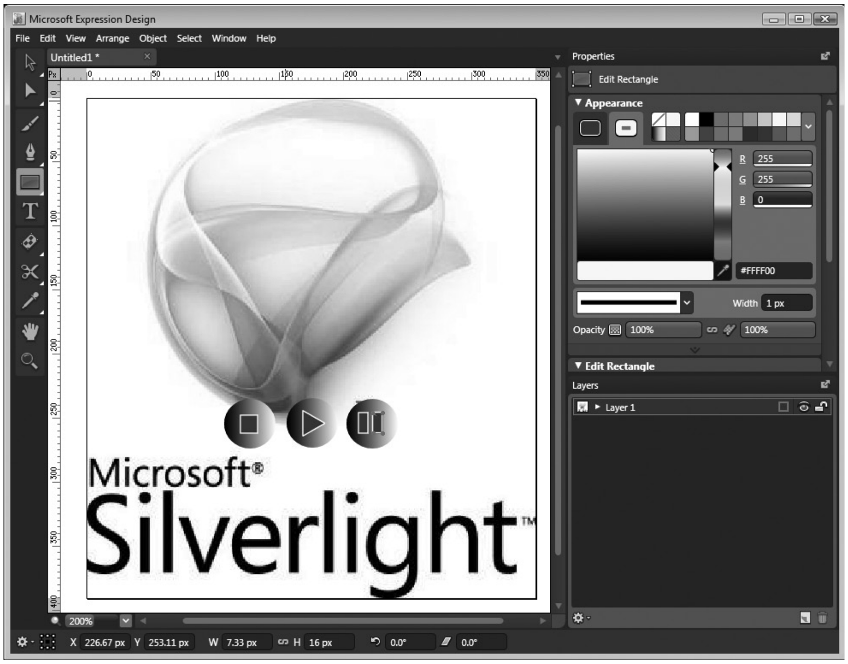
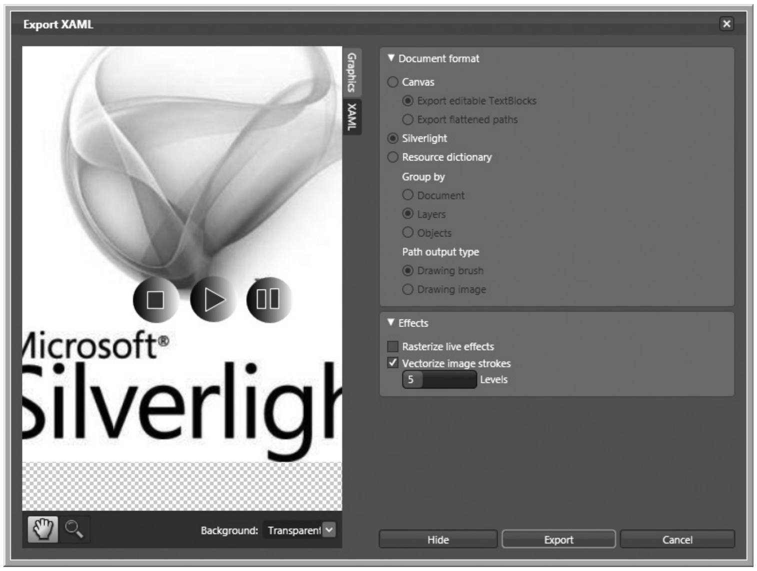


Figure 1-12 Adding the Stop, Play, and Pause buttons.

You're now ready to export your creation as XAML and use it within Expression Blend to implement the application. To do this, select File Export and then select the name that you want to use for the file. The Export dialog box (Figure 1-13) will appear. Make sure that you select Silverlight as the export type.



▼ Document format
- ( ) Canvas
  - (●) Export editable TextBlocks
  - ( ) Export flattened paths
- (●) Silverlight
- ( ) Resource dictionary

Group by
- ( ) Document
- (●) Layers
- ( ) Objects

Path output type
- (●) Drawing brush
- ( ) Drawing image

▼ Effects
- ☐ Rasterize live effects
- ✓ Vectorize image strokes
- 5 Levels

Figure 1-13 Exporting the Video Player application.

Click Export to create the XAML. In addition, a subdirectory will be created with a copy of the .png file in it. In the next section, you'll use this XAML within a new Expression Blend project.

Implementing the Application in Expression Blend

Microsoft Expression Blend allows you to create a Silverlight 1.0 application, as you learned earlier in the chapter. Launch Blend and select New Silverlight 1.0 Script Application from the File/New menu options.

This will create a new Expression Blend project containing the files described earlier in this chapter. You can add the XAML by right-clicking on the project in the project Explorer window and then selecting Add New Item. You'll see the three control icons, but not the background graphic, as this was exported into a subdirectory of the directory where you saved the XAML file. Repeat the process to add a new item and browse to the graphic.

You'll notice if you look at the XAML view that the picture is implemented using the *Image* element. This has a *Source* attribute that contains the subdirectory the image that you created using Design. Since you have imported the image to your Blend project, it's now in the same directory as the XAML, and thus cannot be found. So, edit the *Source* element to remove the subdirectory and set the source attribute to the name of the image (i.e., Source="image0.png"). Now you'll see the image in Blend.

Adding Video Using the Media Element

Silverlight provides audio and visual media using the *MediaElement* control. This isn't present on the toolbar by default. You can add this by clicking the New Tools icon at the bottom of the toolbar to open the Assets Library dialog box, shown in Figure 1-14.

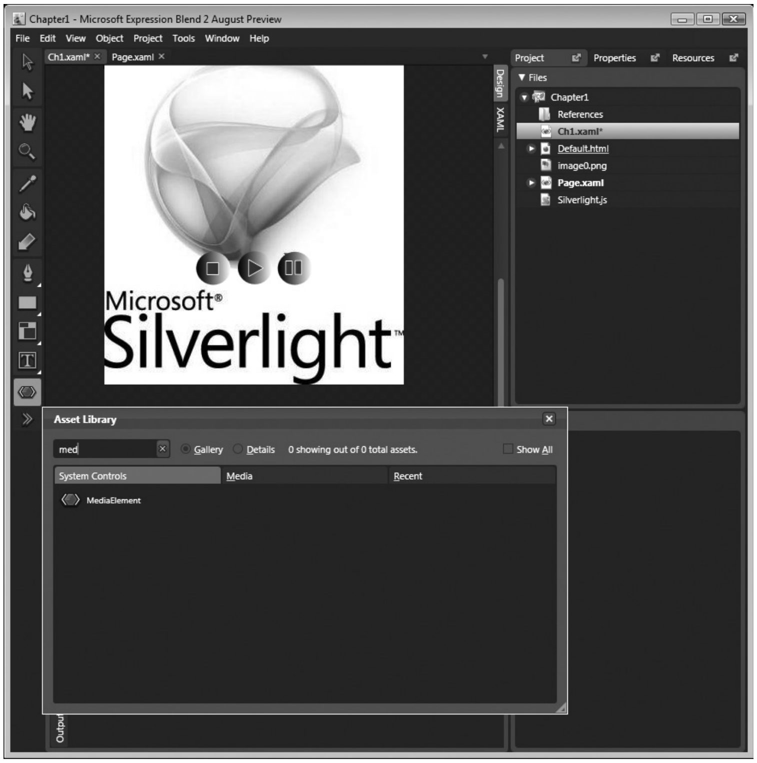


Figure 1-14 Adding a new control using the Asset Library.

You can search the Asset Library using the text box at the upper left-hand corner of the dialog box. Type **Media** (as shown in Figure 1-14), and you'll see the *MediaElement* listed on the System Controls list. Select it and then double-click it to add it to the toolbar.

Next, add the media element to the scene by selecting its tool on the toolbar and dragging out the shape that you want it to cover. You'll see a grey rectangle with a camera icon in the center as a placeholder for the media. See Figure 1-15.

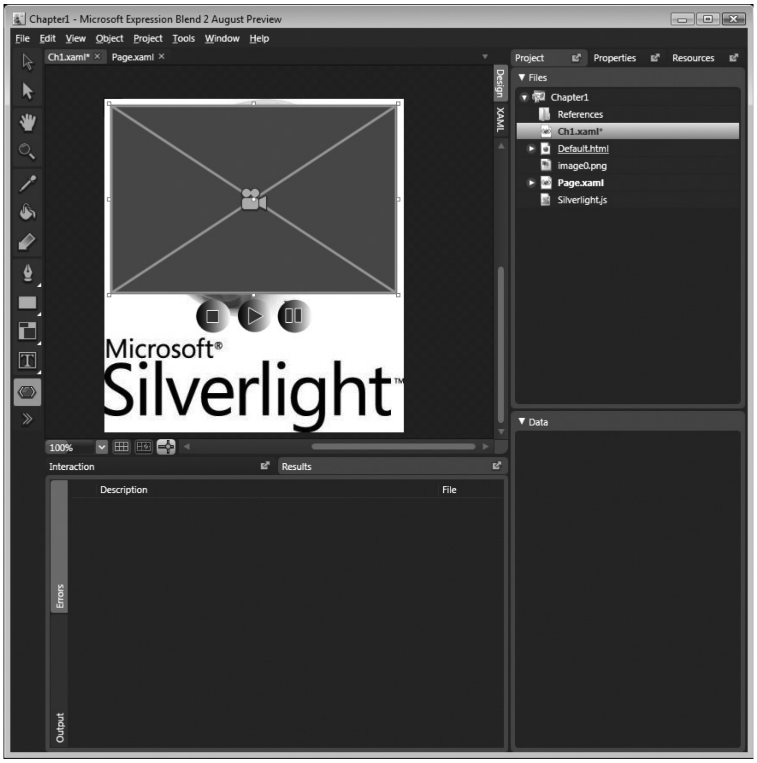
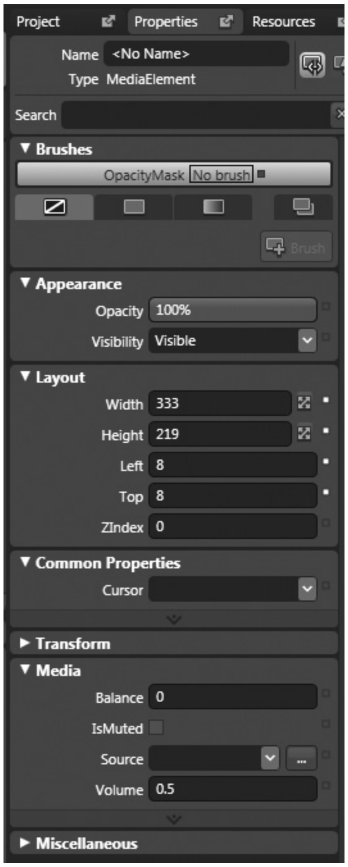


Figure 1-15 Adding a new *MediaElement* to your design.

On the right-hand side of the screen, you can see the Properties dialog box that provides options you can use to configure this video element, as shown in Figure 1-16.



Name <No Name>
Type MediaElement
Search

Opacity 100%
Visibility Visible

Width 333
Height 219
Left 8
Top 8
ZIndex 0

Cursor

Balance 0
IsMuted
Source
Volume 0.5

Figure 1-16 The *MediaElement* Properties dialog box.

You can search the properties using the search box near the top of the dialog box. This is useful for complex objects that have a lot of properties associated with them. To set the video source, use the *Source* property and an available WMV file. The file will be added to the project, and the XAML will be updated to configure the media element to indicate the path to the video in its source attribute. In the XAML editor, set the *MediaElement*'s name to "vid". Your XAML will now look like this:

```
<MediaElement Width="408" Height="232" Canvas.Left="48" Canvas.Top="40" Source="Test.wmv"
x:Name="vid"/>
```

If you test the application now by pressing F5, you'll see the Silverlight content rendered in the browser and the video will play back. The next thing to do is to configure the buttons to catch events and to control the media element in response to them.

Wiring the Video Controls

You're now going to wire events to the controls, but since each control is made up of a number of elements (the background circle and the foreground icon), you can group them into a single canvas and then wire the elements to the canvas together to make things easier.

To do this, make sure that the direct selection tool (it looks like a white arrow) is selected on the toolbar, and then hold the Ctrl key as you select the background circle and the foreground icon of the video control. So, for example, for the Stop control icon, select the background circle and the foreground rectangle.

Once you've selected the elements, open the Object menu, then select Group Into, and then select Canvas. This will group the controls into a *Canvas* container. See Figure 1-17.

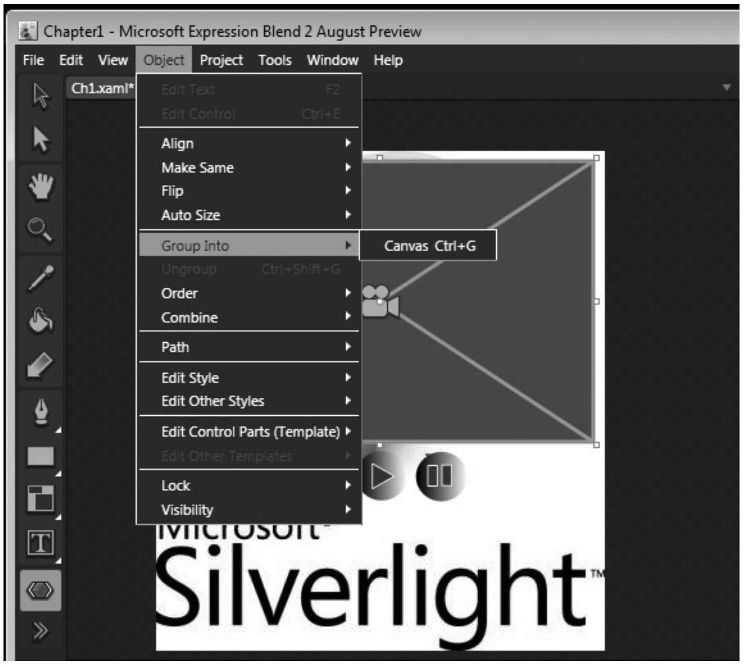


Figure 1-17 Grouping the elements into a *Canvas* container.

The Objects and Timeline dialog box shows the controls in your scene, and you'll now see three new *Canvas* elements that can be opened up to see the contained *Ellipse*, *Rectangle*, and *Path* elements.

It's a good idea, for programmability, to name these *Canvas* elements. Double-click *Canvas* in the Objects and Timeline dialog box and it will be highlighted in yellow. Then you can edit its name using the Properties dialog box.

See Figure 1-18 for an example that shows one *Canvas* element that has been named *btnPlay*, and another *Canvas* element has been selected to edit.

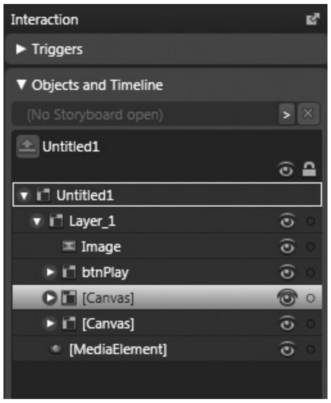


Figure 1-18 Editing the *Canvas* elements.

If you switch to the XAML view, you'll see the *Canvas* elements listed. Following is an example of the XAML for the Play button, which was named *btnPlay*:

```
<Canvas x:Name="btnPlay" Width="52.667" Height="52.667"
        Canvas.Left="236.111" Canvas.Top="390.333">
<Ellipse x:Name="Ellipse" Width="52.6667" Height="52.6667"
      Stretch="Fill" StrokeThickness="2"
      StrokeLineJoin="Round" Stroke="#FFFFFFFF">
    <Ellipse.Fill>
      <LinearGradientBrush StartPoint="0.0189873,0.5" EndPoint="0.981013,0.5">
        <GradientStop Color="#FFE3D2E3" Offset="0"/>
        <GradientStop Color="#FF000000" Offset="1"/>
      </LinearGradientBrush>
    </Ellipse.Fill>
  </Ellipse>
<Path x:Name="Path" Width="22.3439" Height="25.6458"
      Stretch="Fill" StrokeLineJoin="Round" Stroke="#FFFFFF00"
      Data="F1 M 274.867,415.771L 253.524,403.448L 253.524,428.094L
            274.867,415.771 Z "
      Canvas.Left="16.913" Canvas.Top="12.615">
    <Path.Fill>
      <LinearGradientBrush StartPoint="-0.0223775,0.500001"
                           EndPoint="1.02238,0.500001">
        <GradientStop Color="#FFAE69AE" Offset="0"/>
        <GradientStop Color="#FF000000" Offset="1"/>
      </LinearGradientBrush>
    </Path.Fill>
  </Path>
</Canvas>
```

You can see that the *Canvas* element envelopes the *Ellipse* and *Path* elements (the *Path* element implements the triangle shape), so if you define an event handler for the *Canvas* element, it will apply to all of the children of that component, too. So, for example, you can trap the user pressing the mouse button on an element using the *MouseLeftButtonDown* event. If you define this for the *Canvas*, then clicking on the ellipse or on the triangle will raise that event.

Now let's add the event handling for the Play button. On the *Canvas* that represents the Play button, you can add an attribute called *MouseLeftButtonDown*, which contains the name of the desired JavaScript function to execute when the user clicks it. Here's the XAML code with the new attribute highlighted:

```
<Canvas x:Name="btnPlay" Width="52.667" Height="52.667" Canvas.Left="236.111"
Canvas.Top="390.333" MouseLeftButtonDown="handlePlay">
```

Now, all you need to do is create the *handlePlay* function. Open the Default.html.js file and add the following code at the bottom of the existing code:

```
function handlePlay(sender, eventArgs)
{
   sender.findName("vid").play();
}
```

The *findName* function is exposed by the Silverlight DOM API, and it allows you to search for a named element and create a reference to it. In this case, it finds the element called *vid*, which is the *MediaElement* you placed and named earlier. It then calls the *play* method on this to play the video. You can follow a similar approach to wire the *pause* and *stop* methods to their respective *Canvas* elements and write JavaScript that calls the *pause* and *stop* methods, also exposed by the *MediaElement*.

To run and test the application, make sure that the *createSilverlight* function (in Default.html.js) uses the XAML file as its source. If your XAML was called vplayer.xaml, your *createSilverlight* function will look like this:

```
function createSilverlight()
{
   var scene = new Chapter1Project.Scene();
   Sys.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)
      }
   });
}
```

Now you can run the application and control the video playback using the three video controls.

This simple demonstration is intended to show the workflow between Expression Design and Expression Blend, and how they can be used to put together a simple video player application.

It's hard-coded for a single video, and typically, the next step would be to add some new functionality, perhaps to select videos from a list or to give a common dialog box that allowed users to find a video—let your imagination guide you. As you work with Silverlight, ASP.NET, and AJAX, you will see how these technologies all neatly fit together to allow you to build your application the way that you want it, in an easy and productive manner. It should be a fun ride!

Summary

In this chapter, you were introduced to Silverlight and how it fits into the overall Web and user-experience landscape. You discovered how technology from Microsoft is applied to current UX scenarios and were introduced to an overview of the Silverlight architecture.

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.

You took a brief tour of XAML to discover how it is used to implement rich user interfaces, before diving in with both feet and building a simple video player. By the end of this chapter, you should have a better understanding of what it takes to design, implement, and code a user interface in Silverlight 1.0 using JavaScript. 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 Silverlight XAML in the next chapter.

Chapter 5

Programming Silverlight with JavaScript

In this chapter, we'll take an in-depth look at programming the Silverlight object and the XAML it contains using JavaScript within the browser. We'll investigate how to host the Silverlight object in the browser, as well as the full property, method, and event model that the control supports. We'll also look at how to support loading and error events on the control, as well as how to handle parameterization and context for the control. You'll see how Silverlight provides a default error handler and how you can override this with your own error handlers. You'll delve into the *Downloader* object that is exposed by Silverlight, and how this can be used to dynamically add content to your application. Finally, we'll explore the programming model for the user interface (UI) elements that make up the XAML control model, and you will learn how you can use the methods and events that they expose from within the JavaScript programming model.

Hosting Silverlight in the Browser

You don't need any special software to be able to use and build Silverlight applications other than the Silverlight plug-in itself and the Silverlight.js file that manages downloading and installing the plug-in for clients that don't have it. You can use any software for building Web sites to build Silverlight sites, from Notepad to Eclipse to Expression Web or Expression Blend—it's really up to you.

This section presents a basic primer that will show you what you need to do to begin to use Silverlight. So far in this book, you've been using an Expression Blend or Visual Studio template to do the hard work for you, but now let's take a look at what it takes to get a simple Silverlight site up and running without any tools other than Windows Explorer and Notepad.

The first and most important file that you will need is the standard Silverlight.js file. This is available in the Silverlight Software Development Kit (SDK), which you can download from the Web site *http://www.microsoft.com/silverlight*.

Next, you'll need to create an HTML file that will reference this JavaScript file. You'll host the Silverlight control in this page. Here's an example:

```
<HTML>
   <HEAD>
      <script type="text/javascript" src="Silverlight.js" />
   </HEAD>
   <BODY>
   </BODY>
</HTML>
```

The Silverlight.js file contains methods called *createObject* and *createObjectEx* that you can use to instantiate Silverlight. The difference between these is that *createObjectEx* can use the JavaScript Object Notation (JSON) to serialize the parameters.

These functions take a set of parameters that are used to instantiate the control. The parameters are described in Table 5-1.

Table 5-1 Parameters for *createObject* and *createObjectEx*

Parameter Name	Description
source	This sets the source for the XAML code that the control renders. It can be a file reference (i.e.,"source.xaml"), a URI (i.e., *http://server/generatexaml.aspx*) ,or a reference to inline XAML contained within a DIV (i.e., *#xamlcontent* for a DIV named *xamlcontent*).
parentElement	This is the name of the DIV that contains the Silverlight control on your HTML page.
ID	This is the unique ID that you assign to an instance of the Silverlight control.
width	This sets the width of the control in pixels or by percentage.
height	This sets the height of the control in pixels or by percentage.
background	This determines the background color of the control. See the section titled "*SolidColorBrush*" in Chapter 2, "Silverlight and XAML," for more details on how to set colors. You can use an ARGB value, such as #FFAA7700, or a named color, such as *Black*.
framerate	This sets the maximum frame rate to allow for animation. It defaults to 24.
isWindowless	This is set to *true* or *false* and defaults to *false*. When it is set *true*, the Silverlight content is rendered behind the HTML content, so that HTML content can be written on top of it.
enableHtmlAccess	This determines if the content that is hosted in the Silverlight control is accessible from the browser DOM. It defaults to *true*.
inplaceInstallPrompt	Silverlight has two modes of installation. An *inplace* install involves accepting the software license and downloading the control directly without leaving the site hosting it. An *indirect* install involves having the user transfer to the Microsoft download site for Silverlight. From there they accept the license and download the control. You control which method will be presented to the user with this property. Setting the property to *true* allows the user direct *inplace* installation; a *false* setting leads to the *indirect* installation.
version	This determines the minimum version of Silverlight to support.

Table 5-1 **Parameters for *createObject* and *createObjectEx***

Parameter Name	Description
onLoad	This specifies the function to run when the control is loaded.
onError	This specifies the function to run when the control hits an error.
onFullScreenChange	This event is fired when the *FullScreen* property of the Silverlight control changes.
onResize	This event is fired when the *ActualWidth* or *ActualHeight* property of the Silverlight control changes.
initParams	This specifies a user-definable set of parameters to load into the control. For more details on this, see the section titled "Handling Parameters" later in this chapter.
userContext	This specifies a unique identifier that can be passed as a parameter to the *onLoad* event handler function. You'll see more of this in the "Responding to Page Load Events" section later in this chapter.

Please note that the *width*, *height*, *background*, *framerate*, *iswindowless*, *enableHtmlAccess*, *version*, and *inplaceInstallPrompt* properties are handled within a properties array when creating an instance of the control, and the *onLoad* and *onError* are handled within the events array.

Following is an example:

```
Silverlight.createObject(
    "Scene.xaml",
    document.getElementById("SilverlightControlHost"),
    "mySilverlightControl",
    {
         width:'300',
         height:'300',
         inplaceInstallPrompt:false,
         background:'#D6D6D6',
         isWindowless:'false',
         framerate:'24',
         version:'1.0'
     },
     {
         onError:null,
         onLoad:null
     },
     null);
```

This call is typically hosted in an external file, and the de facto standard for this file is named createSilverlight.js. You'll need a reference to this in your HTML to be able to access the function.

So, let's now return to our HTML page and set it up so that it can handle this Silverlight control. You see that when the Silverlight component was named, it was expecting a parent DIV called *SilverlightControlHost* to host it. This is achieved using the *ID* property of the DIV. Here's the full HTML code:

```
<html>
<head>
   <script type="text/javascript" src="Silverlight.js"></script>
   <script type="text/javascript" src="createSilverlight.js"></script>
</head>

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

Finally, you'll need the XAML source for your Silverlight control. This sample calls for a XAML file called Scene.xaml.

Following is a simple XAML file that contains a "Hello, World!" *TextBlock*:

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <TextBlock>Hello, World!</TextBlock>
</Canvas>
```

And that's everything that you need to get a Silverlight application set up and ready to go. As you add more functionality to your application, your code will become more complex, but these four files—the HTML host, the XAML file, createSilverlight.js, and Silverlight.js—are common to every project.

Responding to Page Load Events

You specify a JavaScript event handler to manage page load events using the *onLoad* parameter introduced in Table 5-1. This fires after the XAML content within the Silverlight control has completely loaded. Note that if you have defined a loaded event on any XAML UI element, those events will fire before the Silverlight control's *onLoad* event does. In addition to this, the control has a read-only *IsLoaded* property that is set immediately before the *onLoad* event fires.

When using an *onLoad* event handler, your JavaScript function should take three parameters: The first is a reference to the control, the second is the user context, and the third is a reference to the root element of the XAML. Following is an example:

```
function handleLoad(control, userContext, rootElement)
{
   ...
}
```

Handling Parameters

When you call the *createObject* function to instantiate Silverlight, you can pass parameters to it using the *initParams* property. This property is a string value, so, if you have multiple values, you can encode them into a comma separated string which is easily sliced in JavaScript. Following is an example of setting up the Silverlight control that has three parameters:

```
function createSilverlight()
{
   Silverlight.createObject(
       "Scene.xaml",
       document.getElementById("SilverlightControlHost"),
       "mySilverlightControl",
       {
           width:'300',
           height:'300',
           inplaceInstallPrompt:false,
           background:'#D6D6D6',
           isWindowless:'false',
           framerate:'24',
           version:'1.0'
       },
       {
           onError:null,
           onLoad:handleLoad
       },
       "p1, p2, p3", // Parameter List
       null);
}
```

Here the parameters *p1*, *p2*, and *p3* are encoded into a comma-separated string. JavaScript has a string *split* method that allows you to split a comma-separated string into an array of values.

Following is an example of an *onLoad* event handler that uses this method to split the parameter list into an array of strings and to display each one in an alert box.

```
function handleLoad(control, userContext, rootElement)
{
     var params = control.initParams.split(",");
     for (var i = 0; i< params.length; i++)
     {
         alert(params[i]);
     }
}
```

User Context

An additional parameter that can be passed to the Silverlight control is the context parameter. This will be included directly in the *onLoad* event as the second parameter, typically named *userContext*. It behaves exactly the same as the previous parameters you've seen, in that it is a property value that can be queried after the control has rendered. Typically, user context is not used for control parameters, however. It is instead used as a reference variable to distinguish different controls, though there is nothing to prevent you from using it to parameterize your control.

Following is an example of a page that hosts two Silverlight controls. Note that the name and context variables are set in this host page and then referenced within the JavaScript that creates the Silverlight control:

```
<html>
<head>
   <script type="text/javascript" src="Silverlight.js"></script>
   <script type="text/javascript" src="createSilverlight.js"></script>
   <script type="text/javascript">
       function handleLoad(control, userContext, rootElement)
       {
            alert(userContext);
       }
   </script>
</head>

<body>
   <div id="firstControl">
      <script type="text/javascript">
          var parentElement = document.getElementById("firstControl");
          var name = "agc1";
          var context = "the first control";
          createSilverlight();
      </script>
   </div>
   <div id="secondControl">
      <script type="text/javascript">
          var parentElement = document.getElementById("secondControl");
          var name = "agc2";
          var context = "the second control";
          createSilverlight();
      </script>
   </div>
</body>
</html>
```

Here you can see two DIV elements, called *firstControl* and *secondControl*. They set up the *parentElement*, *name*, and *context* values before calling a modified version of *createSilverlight*:

```
function createSilverlight()
{
    Silverlight.createObject(
        "Scene.xaml",
        parentElement,
        name,
        {
            width:'300',
            height:'300',
            inplaceInstallPrompt:false,
            background:'#D6D6D6',
            isWindowless:'false',
            framerate:'24',
            version:'1.0'
        },
        {
            onError:null,
            onLoad:handleLoad
        },
        "p1, p2, p3", // Parameter List
        context);
}
```

This takes the *parentElement*, *name*, and *context* values that were set up within the HTML page that you saw earlier, demonstrating that the same *createSilverlight* function can be spread across multiple controls.

The HTML page contains the *onLoad* event handler, which you can see here:

```
<script type="text/javascript">
    function handleLoad(control, userContext, rootElement)
    {
         alert(userContext);
    }
</script>
```

This takes the *userContext* parameter and displays the context value in an alert box. You can see how this appears on screen in Figure 5-1, where the HTML page is displaying the context for the first control.

Responding to Page Error Events

Silverlight provides several methods for error handling, depending on the type of error. Errors are raised when the XAML parser hits a problem, loading isn't completed properly, run-time errors are encountered, and when event handlers defined in the XAML document do not have a JavaScript function associated with them.

When initializing a control using the *onError* event handler, you specify a JavaScript function that will be called when an error occurs. However, if you do not specify one (or if you specify it as *null*), the default JavaScript event handler will fire.

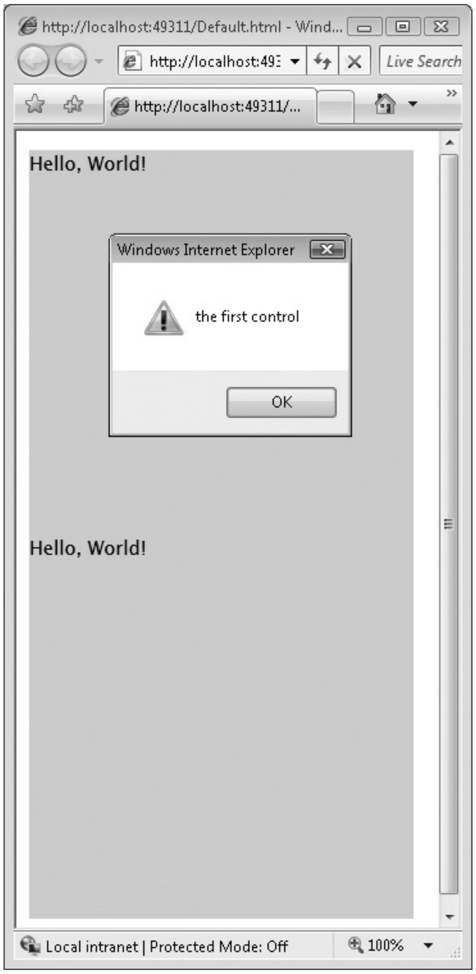

http://localhost:49311/Default.html - Wind...
http://localhost:493
Live Search
http://localhost:49311/...
Hello, World!
Windows Internet Explorer
the first control
OK
Hello, World!
Local intranet | Protected Mode: Off
100%

Figure 5-1 Displaying control context.

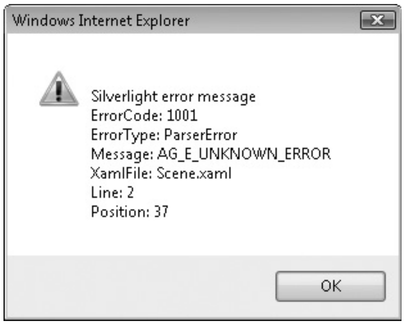
The Default Event Handler

The JavaScript default event handler will display an error message alert box that contains basic details about the Silverlight error, including the error code and type as well as a message defining the specific problem and the method name that was called.

Following is an example of a badly formed XAML document, in which the closing tag of the *TextBlock* element is misnamed *</TextBlok>*:

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <TextBlock>Hello, World!</TextBlok>
</Canvas>
```

If the error handler is set to *null*, then the default error handler will fire and display the default Silverlight error message, as shown in Figure 5-2.



```
Silverlight error message
ErrorCode: 1001
ErrorType: ParserError
Message: AG_E_UNKNOWN_ERROR
XamlFile: Scene.xaml
Line: 2
Position: 37
```

Figure 5-2 The default error message.

Using Your Own Error Handler

You can use your own error handler by setting the *onError* property of the Silverlight control to a custom event handler function. Your error handler function will need to take two parameters: the sender object and the event arguments that define the specifics of the error that occurred.

There are three types of event argument that you can receive. The first is the basic *ErrorEventArgs* object that contains the error message type and code. The *errorType* property defines the type of error as a string containing *RuntimeError* or *ParserError*. Based on this information, you can use one of the two associated derived error types.

When processing a parsing error in XAML, the *ParserErrorEventArgs* is available. This contains a number of properties:

- The *charposition* property contains the character position where the error occurred.
- The *linenumber* property contains the line where the error occurred.
- The *xamlFile* identifies the file in which the error occurred.
- The *xmlAttribute* identifies the xml attribute in which the error occurred.
- The *xmlElement* defines the element in which the error occurred.

Run-time errors are defined in the *RuntimeErrorEventArgs* object. This object also contains a number of properties:

- The *charPosition* property identifies the character position where the error occurred.
- The *lineNumber* property identifies the line in which the error occurred.
- The *methodName* identifies the method associated with the error.

In the previous section, you saw a parsing error as trapped by the default error handler. Here's how you could capture the same error with your own error handler. First, you create the *createSilverlight()* method that sets up the error handler:

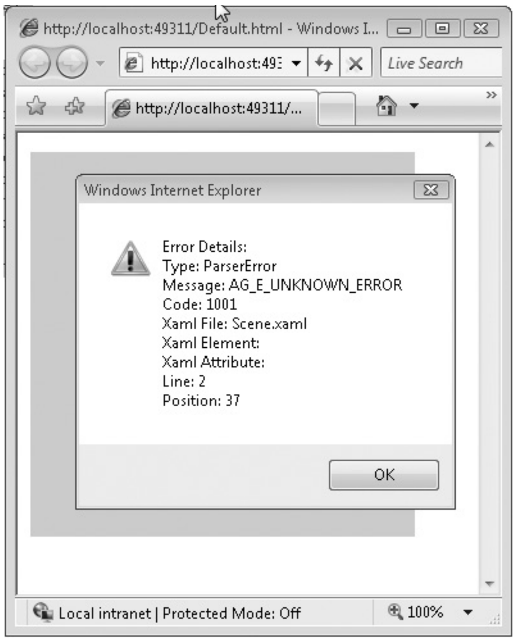
```
function createSilverlight()
{
    Silverlight.createObject(
        "Scene.xaml",
        document.getElementById("firstControl"),
        "agc1",
        {
            width:'300',
            height:'300',
            inplaceInstallPrompt:false,
            background:'#D6D6D6',
            isWindowless:'false',
            framerate:'24',
            version:'1.0'
        },
        {
            onError:handleError,
            onLoad:null
        },
        null);
}
```

Following is the HTML file that calls this revised Silverlight creation method and contains the *handleError* function that was defined as the error handler using the *onError* attribute:

```
<html>
<head>
   <script type="text/javascript" src="Silverlight.js"></script>
   <script type="text/javascript" src="createSilverlight.js"></script>
   <script type="text/javascript">
       function handleError(sender, errorArguments)
       {
           var strError = "Error Details: \n";
           strError+= "Type: " + errorArguments.errorType + "\n";
           strError+= "Message: " + errorArguments.errorMessage + "\n";
           strError+= "Code: " + errorArguments.errorCode + "\n";
           // We know (in this case) that its a parser error.
           // For a more generic error handler
           // you should trap on error type before calling
           // properties on a specific argument type.
           strError+= "Xaml File: " + errorArguments.xamlFile + "\n";
           strError+= "Xaml Element: " + errorArguments.xmlElement + "\n";
           strError+= "Xaml Attribute: " + errorArguments.xmlAttribute + "\n";
           strError+= "Line: " + errorArguments.lineNumber + "\n";
           strError+= "Position: " + errorArguments.charPosition + "\n";
           alert(strError);
       }
   </script>
</head>
```

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

When this is executed and the error is tripped, the alert box will display the contents of the error. Figure 5-3 shows an example of a customized alert box.


http://localhost:49311/Default.html - Windows I...
http://localhost:493
Live Search
http://localhost:49311/...
Windows Internet Explorer
OK
Local intranet | Protected Mode: Off
100%

Error Details:
Type: ParserError
Message: AG_E_UNKNOWN_ERROR
Code: 1001
Xaml File: Scene.xaml
Xaml Element:
Xaml Attribute:
Line: 2
Position: 37

Figure 5-3 Using your own event handler.

Silverlight Control Properties

The Silverlight control has a number of properties, some of which were discussed in the section titled "Hosting Silverlight in the Browser." In addition to being able to set them when you initialize the control, you can also set the controls properties using script. The control splits properties into three types: direct, content, and settings properties. *Direct* properties are properties of the control itself that are accessible using the *control.propertyname* syntax. *Content* properties and *settings* properties are accessed using the *control.content.propertyname* and *control.settings.propertyname* syntax respectively.

Direct Properties

Following are the direct properties that are supported:

- ***initParams*** The initialization parameters that are passed to the control are stored in this property. It can only be set as part of the control initialization.
- ***isLoaded*** This property is *true* after the control is loaded; otherwise it is *false*. It is read-only.
- ***source*** This is the XAML content that you want to render. It can be a reference to a file, a URI to a service that generates XAML, or, when prefixed with a # character, it is a DIV containing XAML code in a script block.

Content Properties

When accessing content properties, you use the *control.content.propertyname* syntax. For example, if you want to access the *actualHeight* property, you use the *control.content.actualHeight* syntax. The following content properties are available:

- ***actualHeight*** This returns the height of the rendering area of the Silverlight control in pixels. The value returned depends on a number of different criteria. First, it depends on how the height of the control was initially set. Recall that it can be a percentage or an absolute pixel value. In the case of the former, the *actualHeight* property is the current height of the control, but if the user changes the browser dimensions, this will change. If the height was set using an absolute pixel value, this will be returned. When the control is used in full screen mode, this will return the current vertical resolution of the display.
- ***actualWidth*** This returns the width of the display. The value returned depends on a number of criteria and is similar to the *actualHeight* parameter.
- ***fullScreen*** This switches the Silverlight control display between embedded and full screen mode. It defaults to *false*, which is the embedded mode. When set to *true*, Silverlight will render to the full screen.

Settings Properties

The control also contains a number of properties that are defined as settings properties, where they are accessed using the *control.settings.propertyname* syntax:

- ***background*** This sets the background color of the Silverlight control. It can take several different formats, including a named color (such as Black), 8Bit Red/Green/Blue (RGB) values with or without alpha, and 16Bit RGB values with or without alpha.
- ***enableFrameRateCounter*** When set to *true*, Silverlight will render the current frame rate (in frames per second) in the browser's status bar. It defaults to *false*.

- ***enableHtmlAccess*** When set to *true*, this will allow the XAML content to be accessible from the browser DOM. The default value is *true*.
- ***enableRedrawRegions*** When set to *true*, this shows the areas of the plug-in that are being redrawn upon each frame. It's a useful tool to help you optimize your application. The default value is *false*.
- ***maxFrameRate*** This specifies the maximum frame rate to render it. It defaults to 24 and has an absolute maximum of 64.
- ***version*** This reports the version of the Silverlight control that is presently being used. It is a string containing up to four integers, separated by dots, which contain the major, minor, build, and revision number, though only the first two values (major and minor version number) are required.
- ***windowless*** This determines whether the property is displayed as a windowless or windowed control. When set to *true*, it is windowless, meaning the Silverlight content is effectively rendered "behind" the HTML content on the page.

Silverlight Control Methods

The Silverlight control has a number of methods that you can use to control its behavior and function. Similar to Silverlight property groups, the Silverlight methods are grouped into "families" of methods. At present, one direct and three content methods are supported. You'll see which is which in the following sections, including samples showing their syntax and how to access them.

The *createFromXaml* Method

The *createFromXaml* method is a Silverlight content method that allows you to define XAML content to dynamically add to your Silverlight control. It takes two parameters. The first is a string containing the XAML that you want to use, and the other is the *namescope* parameter that, when *true* (it defaults to *false*), will create unique *x:Name* references within the provided XAML that will not conflict with any existing XAML element names.

There is a constraint around the XAML that you can add using *createFromXaml*. The XAML you add has to have a single root node. So, if you have a number of elements to add, make sure that they are all contained within a single containing *Canvas* element.

Additionally, *createFromXaml* does not add the XAML to the Silverlight control until it has been added to the children of one of the *Canvas* elements within the control. So, when you call *createFromXaml*, you get a reference to the node returned, and this reference is then used to add the node into the render tree. Following is an example:

```
function handleLoad(control, userContext, sender)
{
   var xamlFragment = '<TextBlock Canvas.Top="60" Text="A new TextBlock" />';
   textBlock = control.content.createFromXaml(xamlFragment);
   sender.children.add(textBlock);
}
```

Here the XAML code for a text block control is created, containing the text "A new TextBlock". This is then used to create an XAML node within the control content, and after it is complete, Silverlight will return a reference to the text block. This reference is then added to the Silverlight control's render tree and is used to render the context of the text block.

The *createFromXamlDownloader* Method

The *createFromXamlDownloader* method is a content method used in conjunction with a *Downloader* object, which you will learn about later in this chapter. It takes two parameters. The first parameter is a reference to the *Downloader* object that downloads the XAML code, or a package containing the XAML code. The second parameter is the name of the specific part of the download content package to use. If this is a .zip file, then you specify the name of the file within the .zip file that contains the XAML code you want to use. When the downloaded content is not in a .zip package, then this parameter should be set to an empty string.

The *createObject* method

The *createObject* method is a direct method designed to allow you to create a disposable object for a specific function. In Silverlight 1.0, the only object that is supported is the *Downloader* object. We'll cover this in greater detail later in this chapter.

The *findName* method

This content method allows you to search for a node within your XAML code based on its *x:Name* attribute. If *findName* finds a node with the provided name, it returns a reference to it; otherwise it returns *null*.

The *Downloader* Object

The Silverlight control provides an object that allows you to download additional elements using asynchronous downloading functionality. This allows you to download individual assets, or assets that are packaged in a .zip file.

Downloader Object Properties

The *Downloader* object supports the following properties:

- ***downloadProgress*** This property provides a normalized value (between 0 and 1) representing the percentage progress of the content downloaded, where 1 is equal to 100 percent complete.

- ***status*** This property gets the HTTP status code for the current status of the downloading process. It returns a standard HTTP status code, for example "404" for "Not Found" or "200" for "OK".
- ***statusText*** This property gets the HTTP status text for the current status of the downloading process. This corresponds to the status code for the *status* property. For a successful request, the *status* will be "200," and the *statusText* will be "OK." For more information about HTTP status codes, check out the standard HTTP codes provided by W3C (*http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html*).
- ***uri*** This property contains the URI of the object that the downloader is presently accessing.

Downloader Object Methods

The *Downloader* object supports the following methods:

- ***abort*** This cancels the current download and resets all properties to their default state.
- ***getResponseText*** This returns a string representation of the downloaded data. It takes an optional parameter that is used to name the contents of the file name within a downloaded package.
- ***open*** This initializes the download session. It takes three parameters. The first is the verb for the action. The set of HTTP verbs is documented by the W3C; however, only the *GET* verb is supported in Silverlight 1.0. The second parameter is the URI for the resource that is to be downloaded. The optional third parameter determines if the download is synchronous or asynchronous. It defaults to *true* (for asynchronous download).
- ***send*** This executes the download request that was initialized with the *Open* command.

Downloader Object Events

The *Downloader* object supports the following events:

- ***completed*** This event will fire when the download is complete. It takes two parameters. The first is the object that raised the event (in this case, the downloader control itself), and the second is a set of event arguments (*eventArgs*). In Silverlight 1.0, the *eventArgs* parameter is always *null*.
- ***downloadProgressChanged*** This event will fire while content is being downloaded. It fires every time the progress (which is a value between 0 and 1) changes by 0.05 (5 percent) or more, as well as when it reaches 1.0 (100 percent). When it reaches 1.0, the *completed* event will also fire.

Using the *Downloader* Object

You create a *Downloader* object using the *createObject* method provided by the Silverlight control. Here's an example:

```
<script type="text/javascript">
    function handleLoad(control, userContext, sender)
    {
       var downloader = control.createObject("downloader");
    }
</script>
```

The next step is to initialize the download session by using the *Downloader* object's *open* method to declare the URI of the file, and then to call the *send* method to kick off the download. Following is an example that will download a movie file called *movie.wmv*:

```
function handleLoad(control, userContext, sender)
{
   var downloader = control.createObject("downloader");
   downloader.open("GET","movie.wmv",true);
   downloader.send();
}
```

In order to trap the download progress and completion, you'll need to wire the appropriate event handlers. Following is the same function, updated accordingly:

```
function handleLoad(control, userContext, sender)
{
   var downloader = control.createObject("downloader");
   downloader.addEventListener("downloadProgressChanged","handleDLProgress");
   downloader.addEventListener("completed","handleDLComplete");
   downloader.open("GET","movie.wmv",true);
   downloader.send();
}
```

Now you can implement these event handlers. In this example, the *DownloadProgressChanged* event is wired to a JavaScript function called *handleDLProgress,* and the *Completed* event is wired to the *handleDLComplete* JavaScript function. You can see these functions here:

```
function handleDLProgress(sender, args)
{
   var ctrl = sender.getHost();
   var t1 = ctrl.content.findName("txt1");
   var v = sender.downloadProgress * 100;
   t1.Text = v + "%";
}

function handleDLComplete(sender, args)
{
    alert("Download complete");
}
```

Programming UI Elements

XAML provides a number of visual elements for creating your user interfaces. These are listed in detail in Chapter 2, where their properties are discussed. The set of UI elements includes *Canvas*, *Ellipse*, *Glyphs*, *Image*, *Line*, *MediaElement*, *Path*, *Polygon*, *Polyline*, *Rectangle*, *Run*, *Shape*, and *TextBlock*.

Each of these elements supports a rich set of methods and events, and these will be listed and discussed in the next several sections.

UI Element Methods

UI elements provide functions that can be called from JavaScript to allow you to manipulate them to create rich application interaction. These methods are common to all of the UI elements.

The *AddEventListener* and *RemoveEventListener* Methods

The *AddEventListener* method is used to add an event listener at run time to the UI element. This is useful for separating design and development—the developer doesn't add anything directly to the XAML that the designer produces. Instead, the developer adds event handling code to a JavaScript file (or block). UI element events you might handle are discussed later in this chapter.

Note When using the *AddEventListener* method, be sure to define the name of the event using a lowercase letter for the first character. For example, if you are defining an event handler for the *MouseLeftButtonDown* event using the *addEventListener* method, you declare it using the string "mouseLeftButtonDown".

Following is an example that shows you how to add an event listener at run time. This example adds an event handler that traps the mouse click and specifies that the event should be handled by a JavaScript function called *handleMouse*. This function, like most event handlers, takes two parameters, a sender and event arguments. Because it is a mouse event, it takes an instance of *MouseEventArgs,* which allows us to get the x- and y-coordinates of the mouse at the time of the event:

```
<script type="text/javascript">
   function handleLoad(control, userContext, sender)
   {
       sender.addEventListener("mouseLeftButtonDown",handleMouse);
   }
   function handleMouse(sender, mouseEventArgs)
   {
       alert(mouseEventArgs.getPosition(null).x + ":"
           + mouseEventArgs.getPosition(null).y);
```

```
    }
</script>
```

You can destroy the connection at run time using the *RemoveEventListener* method with the same syntax.

The *findName* Method

This method is used to search through the child elements of a particular element to find a named object. It will return a reference to the specified object if it exists; otherwise it will return *null*. For example, take a look at this XAML code:

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Height="400" Width="400">
  <TextBlock Canvas.Top="0" x:Name="txt1" Text="TextBlock1" />
  <TextBlock Canvas.Top="20" x:Name="txt2" Text="TextBlock2" />
  <TextBlock Canvas.Top="40" x:Name="txt3" Text="TextBlock3" />
</Canvas>
```

This code defines three text blocks called *txt1*, *txt2*, and *txt3* using the *x:Name* property. You can now use the *findName* method to find a named node, obtain a reference to it, and then edit it using that reference. In this case, it is done within the *handleLoad* event handler.

```
<script type="text/javascript">
    function handleLoad(control, userContext, sender)
    {
         var t1 = sender.findName("txt1");
         t1.Text = "TextBlock1 has changed";
    }
</script>
```

Accessing the Control with the *GetHost* Method

UI elements provide a *GetHost* method that can be used to get a handle to the containing Silverlight control. This is useful when you want to use an event handler on one control to access a different control.

An example of this can be seen in the event handler for the download progress of a *Downloader* object. In this case, the function doesn't have direct access to the Silverlight control, but in order to manipulate the properties of another element, the UI element event handler needs a reference to that element (in this example, a reference to the Silverlight control). The UI element event handler can do this by getting a reference to the control using *getHost*, and then from that reference it can find the other element.

```
function handleDLProgress(sender, args)
{
    var ctrl = sender.getHost();
    var t1 = ctrl.content.findName("txt1");
    var v = sender.downloadProgress * 100;
    t1.Text = v + "%";
}
```

Accessing a Parent Element with the *getParent* Method

You may have cases in which you want to access a UI element's parent. It is inefficient to get a reference to the Silverlight control and then to use *findName* to get the parent, so the *getParent* method is available. It will return a reference to the parent upon success; otherwise it will return *null*.

Using the *GetValue* and *SetValue* Methods

You can always access properties with the traditional dot syntax, *object.propertyname*, but an alternative methodology, using the *GetValue* method, exists to support attached properties. So, for example, if you want to access the *Canvas.Top* property, you cannot do it with *object.Canvas.Top*. You must use the *object.GetValue("Canvas.Top")* syntax. *GetValue* can also be used to access nonattached properties, even though in that case the dot notation is equivalent.

In a similar way, you can use the *SetValue* method to set a property value for either a simple or attached property. This method takes two parameters. The first is the name of the property, and the second is the value to assign. Following is an example:

```
var t1 = ctrl.content.findName("txt1");
t1.setValue("Canvas.Top",20);
```

Using *SetFontSource*

The *TextBlock* element supports an additional method, *SetFontSource*, that adds font files to the object's collection of fonts that it can use. So, if you want to use a new font to render the text—for example, if you need to support a foreign character set (such as a font used for an East Asian language), then you can download the font with a *Downloader* object and use the *SetFontSource* method, passing it the *Downloader* and the *TextBlock* will use that font. To use this method, you must have the rights to distribute the font (or a subset thereof).

Following is an example using the *SetFontSource* method. In this case, I have an XAML document that was defined using Expression Blend and that uses some Chinese text:

```
<Canvas
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="640" Height="480" Background="White">
   <TextBlock x:Name="myTextBlock" Width="152" Height="64"
     Canvas.Left="184" Canvas.Top="56" Text="你好，你好吗?"
     TextWrapping="Wrap" MouseLeftButtonDown="handleIt" />
</Canvas>
```

When this is rendered, the default Silverlight font set will not recognize the Chinese characters and will print them as *unprintable* character blocks (typically small squares). However, you can use a *Downloader* object to download a font that *does* support Chinese text. The previous XAML code defines a *MouseLeftButtonDown* event handler function called *handleIt*. You can see that function here:

```
// Event handler for initializing and executing a font file download request.
function handleIt(sender, eventArgs)
{
   // Retrieve a reference to the control.
   var control = sender.getHost();
   // Create a Downloader object.
   var downloader = control.createObject("downloader");
   // Add Completed event.
   downloader.addEventListener("Completed", "onCompleted");
   // Initialize the Downloader request.
   downloader.open("GET", "SIMHEI.TTF", true);
   // Execute the Downloader request.
   downloader.send();
}
```

This creates a download object that downloads the font and defines an event handler *onCompleted* that will handle the *Completed* event that fires when the download is complete. This event will then set the font source for the *TextBlock* to the supporting font, and then Silverlight will render the Chinese characters using the new font source.

```
// Event handler for the Completed event.
function onCompleted(sender, eventArgs)
{
   // Retrieve the TextBlock object.
   var myTextBlock = sender.findName("myTextBlock");

   // Add the font files in the downloaded object
   // to the TextBlock's type face collection.
   myTextBlock.setFontSource(sender);

   // Set the FontFamily property to the friendly name of the font.
   myTextBlock.fontFamily = "Simhei";
}
```

UI Element Events

UI elements support a number of events that may be wired to JavaScript functions either by using the *AddEventListener* methodology to add them at run time, or by using the appropriate XAML attribute to add them at design time. For example, if you want to wire a control's *MouseLeftButtonDown* event using JavaScript, you do it with the *AddEventListener* method:

```
t1.addEventListener("MouseLeftButtonDown", handleMouseDown);
```

To wire the event in XAML, you can use the attribute that has the same name as the desired event (such as "MouseLeftButtonDown"). Following is an example:

```
<TextBlock Canvas.Top="0" x:Name="txt1" Text="Status"
MouseLeftButtonDown="handleMouseDown"/>
```

The events that are supported on the UI element are as follows:

- ***GotFocus*** This is fired when the element receives mouse focus.
- ***KeyDown*** Occurs on an element when it has focus and a key is pressed. The event handler takes two attributes. The first of these is the *sender*, representing a reference to the object that raised the event. The second is a *KeyEventArgs* object. This has a number of properties of its own. The first is *key*, which is an integer value that represents the key that was pressed. It is not operating-system specific, and specific details about how this maps to actual keys can be found in the Silverlight SDK. Another property is the *platformKeyCode*, which *is* operating-system specific. In addition to the actual key, the Boolean properties *shift* and *ctrl* are exposed. These indicate whether or not the Shift and Ctrl keys are pressed.
- ***KeyUp*** This occurs on an element when it has focus and the key is released. It provides for the same two attributes as the *KeyDown* event.
- ***Loaded*** This fires when the Silverlight content is loaded into the host Silverlight control and parsed, but before it is rendered.
- ***LostFocus*** This is the opposite of the *GotFocus* event; it fires when the object loses focus.
- ***MouseEnter*** This fires when the mouse enters the bounding area of the object.
- ***MouseLeave*** This is the opposite of *MouseEnter* event; it fires when the mouse leaves the area of the bounding object.
- ***MouseLeftButtonDown*** This occurs when the user presses the left mouse button over the UI element.
- ***MouseLeftButtonUp*** This occurs when the left mouse button is released.
- ***MouseMove*** This occurs when the cursor moves over the UI element.

Implementing Drag and Drop

You can use the mouse event handlers and Silverlight's *CaptureMouse* and *ReleaseMouseCapture* methods to implement drag and drop in Silverlight.

First of all, let's take a look at a XAML document containing several shapes that may be dragged and dropped around the canvas. These shapes wire their mouse event handlers (*MouseLeftButtonDown*, *MouseLeftButtonUp* and *MouseMove*) to the *onMouseDown*, *onMouseUp*, and *onMouseMove* functions respectively.

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Height="400" Width="400">
<Ellipse Canvas.Top="0" Height="10" Width="10" Fill="Black"
         MouseLeftButtonDown="onMouseDown"
         MouseLeftButtonUp="onMouseUp"
         MouseMove="onMouseMove" />
<Ellipse Canvas.Top="20" Height="10" Width="10" Fill="Black"
         MouseLeftButtonDown="onMouseDown"
         MouseLeftButtonUp="onMouseUp"
         MouseMove="onMouseMove"/>
<Ellipse Canvas.Top="40" Height="10" Width="10" Fill="Black"
         MouseLeftButtonDown="onMouseDown"
         MouseLeftButtonUp="onMouseUp"
         MouseMove="onMouseMove"/>
<Ellipse Canvas.Top="60" Height="10" Width="10" Fill="Black"
         MouseLeftButtonDown="onMouseDown"
         MouseLeftButtonUp="onMouseUp"
         MouseMove="onMouseMove"/>
</Canvas>
```

Now let's take a look at each of the event handler functions. First, let's examine the mouse down event handler. When dragging, you want that control to "own" the mouse events, so you use the *captureMouse* method. You'll also want to remember the starting points for the dragging, so these will be recorded by the event handler. Finally, you'll flag that the mouse is down using a Boolean variable *isMouseDown*.

```
var beginX;
var beginY;
var isMouseDown = false;
function onMouseDown(sender, mouseEventArgs)
{
    beginX = mouseEventArgs.getPosition(null).x;
    beginY = mouseEventArgs.getPosition(null).y;
    isMouseDown = true;
    sender.captureMouse();
}
```

Now, in a drag-and-drop operation, you want to move the item with the mouse. So, when the *MouseMove* event fires, you will record the current mouse coordinates and use them to figure out where the item should be moved as well.

The *mouseEventArgs* allow us to retrieve the current x- and y-coordinates of the mouse, and since the *Ellipse* object that is being dragged in the example is the *sender*, you can set its left and top properties by adding the delta on the x- and y-coordinates to their respective initial values.

Also note that *onMouseMove* will fire whether you are dragging or not, so we use the *isMouse-Down* to check if we are currently dragging. (Remember, it was set in the previous *MouseDown* event handler.)

```
function onMouseMove(sender, mouseEventArgs)
{
  if (isMouseDown == true)
    {
        var currX = mouseEventArgs.getPosition(null).x;
        var currY = mouseEventArgs.getPosition(null).y;
        sender["Canvas.Left"] += currX - beginX;
        sender["Canvas.Top"] += currY - beginY;
        beginX = currX;
        beginY = currY;
    }
}
```

Finally, when the mouse button is released, you will release the mouse capture and reset *isMouseDown*. The ellipses will stay in their new positions.

```
function onMouseUp(sender, mouseEventArgs)
{
    isMouseDown = false;
    sender.releaseMouseCapture();
}
```

Figure 5-4 shows the four ellipses with drag and drop enabled, and Figure 5-5 shows the position of the ellipses after the drag-and-drop operation has been completed.

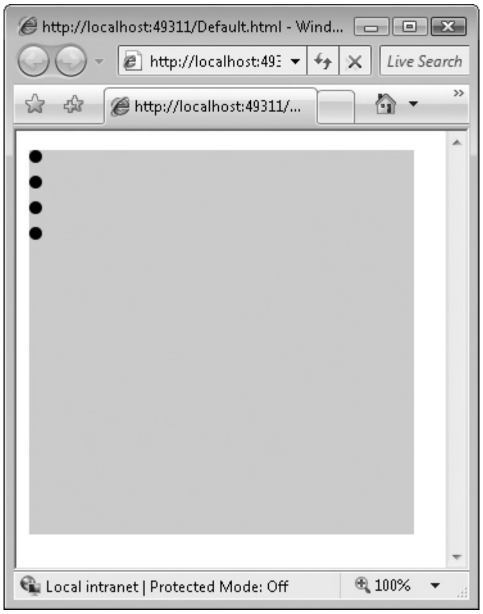

http://localhost:49311/Default.html - Wind...
http://localhost:493
Live Search
http://localhost:49311/...
Local intranet | Protected Mode: Off
100%

Figure 5-4 Four ellipses with drag and drop enabled.

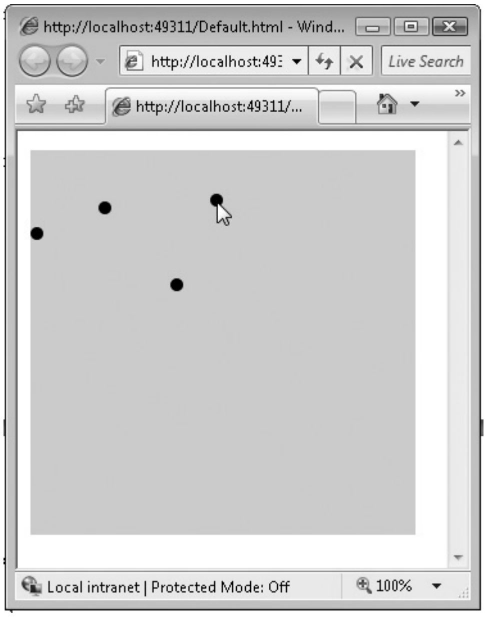

http://localhost:49311/Default.html - Wind...
http://localhost:493
Live Search
http://localhost:49311/...
Local intranet | Protected Mode: Off
100%

Figure 5-5 Dragging and dropping the ellipses.

Summary

In this chapter, you looked at the Silverlight object model and how it can be programmed using JavaScript. You learned how the Silverlight control is hosted within the browser, including how to initialize it, and how to set up its initial state using its property model. You saw how it can be customized with context and custom parameters, as well as the full property, method, and event model that it supports. In addition, you learned how to implement a custom error handler on the control, as well as how to use the basic default error handler.

You discovered how you can add external content to your control using the *Downloader* object and how to trap the events that it exposes to provide feedback as to download progress. You were introduced to the UI elements that XAML offers and learned about the methods, events, and properties that they expose and how you can program them.

In the next chapter, you'll look at how Silverlight supports ink and find out how the Tablet PC, the Touch Screen PC, and other interactive user interface types may be supported by your Web applications using Silverlight.

INTRODUCING MICROSOFT® SILVERLIGHT™ 1.0

Your first look at building rich interactive applications using Silverlight.

Get a head start using Silverlight—the cross-platform, cross-browser plug-in for developing next-generation applications. This practical introduction delivers advance insights and extensive code samples—simplifying the way you implement compelling user experiences for the Web.

Discover how to:

> Use XAML to define layouts, complex shapes, and other visual elements

> Implement transforms and animations—bring your user interface to life

> Enable audio and video playback, including streaming media and progressive download

> Build interactive applications that support inking with a stylus, mouse, or touchscreen

> Support an object-oriented programming model with JavaScript

> Dynamically generate XAML server side from ASP.NET, J2EE, and PHP applications

> PLUS—get a look at Silverlight 1.1, including managed code and dynamic language features

Get XAML and JavaScript code samples on the Web

*For **system requirements**, see the Introduction.*

About the Author:

Laurence Moroney is a Technology Evangelist at Microsoft, specializing in Silverlight and user experience. He has more than a decade of experience in software development and architecture, and is the author of several books on Microsoft ASP.NET, Windows® Presentation Foundation, Web services security, and interoperability.

ISBN-13: 978-0-7356-2539-6

9 780735 625396 90000

Part No. X14-14868

U.S.A. $34.99

[Recommended]

Programming/
Windows/Web

Microsoft® Silverlight™

Microsoft®