CHAPTER 1

# Getting Started

Despite all the wonderful things you can say about HTML, CSS, and JavaScript, I think most people doing a lot of web-based development would agree that they form a pretty poor environment for developing modern sites and applications. If you care about your content working on most web browsers (or even just Internet Explorer and Firefox), accommodating their differences can be maddening. Many techniques and JavaScript libraries have been developed and shared over the years that can reduce this frustration, but none of them are silver bullets.

In addition to browser differences, the graphical capabilities of HTML are too limiting for many user experiences that people want to create. Drawing a simple line, incorporating video, and a number of other things are extremely difficult or impossible with HTML alone. It's not that these technologies were poorly designed, but simply that they were designed for hyperlinked documents rather than the extremely rich presentations that most people want to create on the Web these days.

Considering these issues, it's no wonder that Adobe Flash has been so successful. Whether someone wants to create a professionally designed website, an online game (or any number of other applications), or even a simple advertisement, Flash has been a natural choice for escaping the limitations of HTML. If you doubt the pervasiveness of Flash, try this experiment: Think of a brand of food you eat, and then navigate to the brand's website. Chances are you'll find Flash content at your destination. (I just tried pepsi.com, doritos.com, and oscarmayer.com, and all three are using Flash at the time of writing.) The Flash development experience leaves much to be desired, however. Flash

## IN THIS CHAPTER


- Embedding the Silverlight Control Manually
- Letting `Silverlight.js` Handle the Dirty Work
- Understanding Your Hosting Options
- Interacting with the Silverlight Control Programmatically

(the runtime environment, as well as the tool) suffers from the same basic problem as HTML: Many people are trying to use it for creating rich applications, but it was originally designed for something else (in this case, simple animations).

This is why the introduction of Silverlight is so exciting. A promising alternative to Flash, Silverlight enables the creation of rich web content and applications using a lightweight add-on that is friendly to both designers *and developers*. Yes, the first version of Silverlight is primitive in areas, but it's a true development platform based on concepts and APIs introduced with Windows Presentation Foundation (WPF) in 2006 and in development for many years prior. And, unlike just about any software that has come out of Microsoft, Silverlight is a small download! Version 1.0 is less than 1.5MB, so users who don't have it can get it pretty quickly when browsing to Silverlight content. (By default, Silverlight also automatically updates to later versions when they are available.) Silverlight might just be the silver bullet many designers and developers have been waiting for.

Silverlight 1.0 applications are created with a mixture of XAML (Extensible Application Markup Language), HTML, and JavaScript, so they are easy to integrate into existing web content and compatible with popular Asynchronous JavaScript and XML (AJAX) libraries and techniques. XAML is an XML-based declarative language described in depth in the next chapter. In typical Silverlight applications, a XAML file contains a hierarchy of visual elements that must be rendered on the screen. Silverlight parses the XAML content on initialization, and then renders the content as appropriate.

## DIGGING DEEPER

### A Note for Those Afraid of JavaScript

A few readers might be excited at the idea of using JavaScript to create Silverlight content or applications. If you're like most developers I know, however, you're disappointed to be "forced" to use it in version 1.0. However, programming in JavaScript isn't the worst thing in the world. JavaScript is a very powerful dynamic language, and you can even use it in an object-oriented way if you follow clever patterns that people have devised over the years. (Note that JavaScript really has nothing to do with Java.)

In addition, now that Asynchronous JavaScript and XML (AJAX) is all the rage, there are a number of useful tools and libraries to help you be productive with JavaScript, and they keep getting better. Visual Studio 2008 boasts a number of improvements for JavaScript development, especially related to debugging and IntelliSense.

The pain of programming in JavaScript (when used as part of a website) is often not because of the language itself but rather differences in the HTML Document Object Model (DOM) provided by various web browsers. Fortunately, writing JavaScript that interacts solely with Silverlight objects doesn't have this issue because the Silverlight object model remains the same regardless of the host browser. Most Silverlight applications still require JavaScript that interacts with the HTML DOM, but your exposure to the DOM can be much more limited. And for those cases, ASP. NET AJAX (or other popular AJAX libraries) is a good fit for hiding browser differences.

**Continued**

If you're still not convinced, rest assured that the next version of Silverlight (already available in prerelease form) supports procedural code written in C#, Visual Basic, IronRuby, IronPython, and other .NET languages. And for those who love JavaScript, the next version of Silverlight should support compiled (.NET-based) JavaScript, giving performance that is orders of magnitude faster than the interpreted JavaScript running in browsers today. Some of these languages will be part of the core Silverlight download, whereas other languages might require additional on-demand downloads.

## FAQ

### What are the differences between Silverlight and Adobe Flash?

"Flash" is the name for both a runtime component and a design tool. "Silverlight" refers to a runtime component only, but there are both design tools (such as Expression Blend) and development tools (such as Visual Studio) for Silverlight.

For years, Flash has been the only viable option for rich web-based content, and now Silverlight is positioned to fill the same need. The two technologies have similar features, but there are naturally pros and cons to each.

The biggest thing Flash (the runtime component) has going for it is ubiquity. A website can use Flash with confidence that the vast majority of viewers already have the necessary player installed. Silverlight, on the other hand, is brand new and will take some time to spread—dependent on the amount of compelling Silverlight content out in the wild. Of course, both Flash and Silverlight are designed to have a quick and painless installation, so sites don't have to inconvenience users *too* much if they don't have the necessary software. But even if Silverlight spreads like wildfire during the first few months, the Flash runtime component can still reach places that Silverlight can't (yet), such as mobile devices.

Flash has a variety of visual features that Silverlight lacks, such as bitmap effects (blurring and glowing) and shape tweening (morphing the shape of an object in an animation). Notable features of Silverlight that Flash lacks are higher quality video (even HD 720p full-screen with reasonable hardware) with VC-1 codecs included, seamless interaction with HTML, support for high-resolution and pressure-sensitive input data from a stylus or touch device, and content that's more discoverable to search engines by default thanks to the use of XML rather than compiled script.

The biggest advantage of Silverlight over Flash is in the design of the platform and its associated tools. This advantage becomes especially apparent if you're building an interactive application rather than a simple piece of content. Flash (the design tool) has a huge learning curve for creating an application with even a small amount of logic, and the resulting code is often quite unnatural (and hard to debug). But most software developers, or even people who dabble with HTML and a little bit of programming, should find the learning curve for Silverlight to be pretty small. And if you happen to already be familiar with WPF, learning Silverlight is a breeze.

## FAQ

### What are the differences between Silverlight and WPF?

Whereas Silverlight is designed for creating rich web content or applications that can be viewed in multiple browsers and multiple operating systems, WPF is designed for creating rich Windows applications. WPF applications require the .NET Framework 3.0 or later, which is a much larger download than Silverlight, although Windows Vista and later operating systems already have it installed by default.

Silverlight 1.0 is essentially a subset of WPF, although Silverlight also has a few unique pieces related to video, on-demand downloading of any content, and the control that hosts the content inside a web page. Some WPF features missing from Silverlight 1.0 are common user interface controls (such as buttons and scrollbars), layout panels, 3D graphics, data binding, rich document support, performance optimizations from hardware accelerated graphics, and more. In addition, Silverlight 1.0 applications don't have the benefit of the depth and breadth of the .NET Framework APIs, unless you use them from server-side ASP.NET code. The next version of Silverlight will close some of the gap between Silverlight and WPF, but it will undoubtedly always remain a subset of what WPF and the full .NET Framework provide.

Although Silverlight 1.0 coding is done in JavaScript, which is a big departure from the .NET languages used with WPF (and future versions of Silverlight), the two technologies are highly compatible. In some cases, Silverlight code related to user interface—especially XAML content—can be reused in WPF applications with little work, and vice versa. The key to choosing between Silverlight and WPF is whether you want to optimize for reach or for rich functionality. This is really no different than the classic choice of going with a web application or a Windows (or other OS) application. Besides aforementioned features such as 3D graphics, WPF applications are a natural choice if you require offline support or extensive local storage.

WPF doesn't only support Windows applications, but also applications that run inside the browser called XAML Browser Applications (XBAPs). XBAPs can arguably be considered web applications because their content renders seamlessly inside the browser similar to Silverlight content. However, XBAPs require the .NET Framework 3.0 or later, so they only run on Windows (and only then if the .NET Framework is installed) and only work inside Internet Explorer and Firefox. (Furthermore, Firefox support requires the .NET Framework 3.5 or later.) XBAPs support a much larger subset of WPF functionality than Silverlight 1.0, so they can be an appropriate choice for creating very rich applications that are web-like in their deployment. For example, the British Library has an application called "Turning the Pages" (at http://ttpdownload.bl.uk/browserapp.xbap) that takes advantage of WPF 3D graphics inside the browser.

## FAQ

### What is the relationship between Silverlight 1.0 and the prerelease version of Silverlight?

It's a bit unusual that the next version of Silverlight (currently labeled 1.1) has been available in a prerelease form before Silverlight 1.0 was even finished, but as the version number suggests, it simply is the next version of Silverlight. This next version is a superset of Silverlight 1.0 and is still a subset of WPF and the .NET Framework (but with some unique features of its own). The most notable additions planned for the next version of Silverlight are

- .NET support, which not only means additional language support, but also a subset of the .NET Framework's base class libraries
- Several features that already exist in WPF: user interface controls, layout, data binding, and more
- Potential support for additional browsers and additional operating systems (such as Windows 2000)

Despite all this, everything you learn about Silverlight 1.0 is directly applicable to future versions of Silverlight.

## FAQ

### What web server is required for serving Silverlight content?

Any web server will do, although be sure to set up the MIME type for `.xaml` files. Using Windows Server can give additional benefits when it comes to streaming media, such as the Faststream technology in Windows Media Services. Silverlight Streaming by Windows Live (http://streaming.live.com) can also be an attractive option for hosting Silverlight content on someone else's web server. It supports scalable streaming free (if you don't mind advertisements being served with your content) or for a small fee.

## FAQ

### What are the differences between Silverlight for Windows, Silverlight for Mac OS X, and Silverlight for Linux?

Silverlight supports the same feature set, because it is designed to be completely compatible between all the operating systems and browsers it supports. One advantage Silverlight has on Windows is the ability to get high-resolution and pressure-sensitive input data from a stylus or touch device, although this extra information is given in a way that avoids the need to write Windows-specific code. (See Chapter 7, "Responding to Input Events," for more details.) Silverlight also has different performance characteristics on different browsers and operating systems. For example, windowless controls (described later in the chapter) and elements with transparency are especially slow in Safari on Mac OS X. And of course, Silverlight has bugs that only apply to a specific browser or operating system. Some of these are pointed out in this book.

## Embedding the Silverlight Control Manually

Silverlight, just like Adobe Flash, is a web browser add-on. It's a pair of components—one for Internet Explorer (an ActiveX control), and one for all other supported browsers (a Netscape plug-in)—but this is an invisible implementation detail to make things "just work" regardless of the host browser. The standard way for web pages to take advantage of an add-on—whether Silverlight, Flash, or another—is with the `OBJECT` HTML element.

Listing 1.1 contains a simple web page for a fictional "Great Estates" housing development that embeds a Silverlight logo at the top using the `OBJECT` element.

LISTING 1.1 A Web Page with Embedded Silverlight Content

```
<html>
  <head>
    <title>Great Estates</title>
  </head>
  <body style="background:blue">
    <!– A Silverlight-based logo: –>
    <object type="application/x-silverlight" id="silverlightControl"
      width="390" height="100">
      <param name="background" value="Yellow"/>
      <param name="source" value="Chapter1.xaml"/>
    </object>
    <p style="font-family:Tahoma; color:white">
      An idyllic new community located high on a hill and offering captivating
      waterfront views. Tailored to meet both the needs of upsizing and
      downsizing buyers, Great Estates offers custom quality architecture and
      design at an affordable price point.
    </p>
  </body>
</html>
```

The `id`, `width`, and `height` attributes on the `OBJECT` element work the same way as on elements such as `DIV`, `TABLE`, and so on. For example, `width` and `height` can be specified in absolute pixel values or as a percentage. The `type` attribute refers to the MIME type of the add-on content. The Silverlight add-on is invoked by the host browser for any content of type `application/x-silverlight`.

The Silverlight add-on supports several custom parameters, covered later in the "Understanding Your Hosting Options" section. In this example, the `background` parameter is set to fill the 390x100 region with the color yellow, and the `source` parameter is pointing to a separate XAML file containing the content to be rendered on top of the yellow background. This XAML file, `Chapter1.xaml`, is shown in Listing 1.2.

LISTING 1.2 `Chapter1.xaml`—A XAML File Containing a Logo

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007">
  <MediaElement Name="video" Source="Lake.wmv" Opacity="0" IsMuted="true"/>
  <!-- A circle containing a live video: -->
  <Ellipse Width="100" Height="100">
    <Ellipse.Fill>
      <VideoBrush SourceName="video"/>
    </Ellipse.Fill>
  </Ellipse>
  <!-- Two pieces of text: -->
  <TextBlock FontFamily="Georgia" Foreground="Blue" FontStyle="Italic"
    FontSize="40" Canvas.Left="125" Canvas.Top="20" Text="Great Estates"/>
  <TextBlock Foreground="Blue" Canvas.Left="110" Canvas.Top="70"
    Text="Luxurious Living at an Affordable Price"/>
  <!-- Curves and a line: -->
  <Path Stroke="Red" StrokeThickness="4">
    <Path.Data>
      <PathGeometry>
        <PathFigure StartPoint="0,65">
          <ArcSegment SweepDirection="Clockwise" Size="2,2" Point="25,65"/>
          <ArcSegment SweepDirection="Clockwise" Size="2,2" Point="50,65"/>
          <ArcSegment SweepDirection="Clockwise" Size="2,2" Point="75,65"/>
          <ArcSegment SweepDirection="Clockwise" Size="2,2" Point="100,65"/>
          <LineSegment Point="390,65"/>
        </PathFigure>
      </PathGeometry>
    </Path.Data>
  </Path>
</Canvas>
```

This XAML file defines a logo containing two lines of text, some vector artwork, and even a live video cropped by a circle! Don't worry about the syntax of the XAML file for now. The next chapter covers everything you need to know about XAML syntax, and the various Silverlight elements (`Canvas`, `MediaElement`, `Ellipse`, and so on) are covered throughout the remainder of the book.

Figure 1.1 displays the web page defined by Listings 1.1 and 1.2. Most web pages probably would make the Silverlight


FIGURE 1.1 Silverlight content manually hosted in a web page with the `OBJECT` element.

content blend in better by giving the OBJECT element a matching background, but for this example, the yellow background helps to highlight the area of the page rendered by Silverlight.

Of course, the Great Estates web page only resembles what's shown in Figure 1.1 if the viewer has the Silverlight add-on installed. Without the add-on, the page looks similar to Figure 1.2 (depending on which browser you use).

Fortunately, there's a relatively easy solution for giving users who don't have the add-on a reasonable experience. If you place content directly inside the OBJECT element, browsers will render that content in the case of failure. Therefore, the OBJECT element in Listing 1.1 could be updated as follows to downgrade the logo to a simple image for viewers without Silverlight:

```
<object type="application/x-sil-
verlight" id="silverlightControl"
  width="390" height="100">
  <param name="background"
value="Yellow"/>
  <param name="source"
    value="Chapter1.xaml"/>
  <!-- Alternative content: -->
  <img src="logo.png"/>
</object>
```

**WARNING**

**HTML and CSS fonts, colors, and more are not inherited by Silverlight content!**

The fonts, colors, and other visual aspects of Silverlight content are completely independent from any other settings on the page. If you want to apply different themes to your Silverlight content, you'll need to employ a custom mechanism to make this happen.

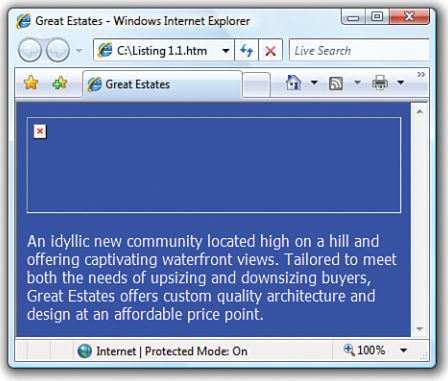


FIGURE 1.2 Listing 1.1 doesn't look good when the Silverlight add-on is missing or disabled.

The logo in logo.png could look identical to the Silverlight logo shown in Figure 1.1, except that the live video would be a static image instead. If you don't want to create a downgraded version of your Silverlight content, you could always notify the user and help her install the Silverlight add-on:

```
<object type="application/x-silverlight" id="silverlightControl"
  width="390" height="100">
  <param name="background" value="Yellow"/>
  <param name="source" value="Chapter1.xaml"/>
  <!-- Alternative content: -->
  This content requires Silverlight. <a href=
    "http://www.microsoft.com/silverlight/downloads.aspx">Get it here.</a>
</object>
```

Unfortunately, Apple's Safari web browser doesn't currently support the OBJECT element. Instead, you must use an element called EMBED, which also happens to work in Internet Explorer and Firefox. Listing 1.3 contains this update to Listing 1.1 in order to work on Safari as well.

LISTING 1.3 Embedding Silverlight Content Using EMBED Instead of OBJECT

```
<html>
  <head>
    <title>Great Estates</title>
  </head>
  <body style="background:blue">
    <!-- A Silverlight-based logo: -->
    <embed type="application/x-silverlight" id="silverlightControl"
      width="390" height="100" background="Yellow" source="Chapter1.xaml"/>
    <p style="font-family:Tahoma; color:white">
      An idyllic new community located high on a hill and offering captivating
      waterfront views. Tailored to meet both the needs of upsizing and
      downsizing buyers, Great Estates offers custom quality architecture and
      design at an affordable price point.
    </p>
  </body>
</html>
```

Besides the different element name (EMBED versus OBJECT), the only other difference is that the custom parameters are specified as attributes of the EMBED element rather than as child elements. Alternative content (for when the embedding fails) can be specified with a separate NOEMBED element. The result from using EMBED looks the same as Figure 1.1 (at least the Silverlight content), as seen in Figure 1.3.


FIGURE 1.3 Silverlight content manually hosted in a web page with the EMBED element, viewed in Apple's Safari browser on Mac OS X.

Using EMBED is the simplest way to get your content rendered in all supported browsers, despite the fact that OBJECT is preferred for Internet Explorer and Firefox.

## Letting Silverlight.js Handle the Dirty Work

Embedding Silverlight content manually with an OBJECT or EMBED element has a number of issues. There's the concern about browser differences (although that can be avoided by always sticking to EMBED). Most importantly, it would be a fair amount of work to properly handle Silverlight detection. For example, although placing a download link as alternative content inside the OBJECT element (or using a NOEMBED element) seems simple

enough, it doesn't behave appropriately if somebody has the *wrong version* of Silverlight installed. If a web page contains Silverlight content that uses future features unavailable in 1.0, viewers with 1.0 installed will not see the alternative content. Instead, the Silverlight 1.0 add-on will attempt to render the content and will fail.

Microsoft would be making a huge mistake if they asked everyone to do the appropriate version detection work on their own. The code involved is not straightforward, and version detection logic—for *any* software—is notorious for being done incorrectly. (As silly as it sounds, someone might write logic that behaves properly for version numbers such as 1.0 and 1.1, but would fail years later when version 4.0 appears.) Sure enough, the Silverlight Software Development Kit (Silverlight SDK) provides a JavaScript file called `Silverlight.js` that defines a simple JavaScript function handling everything from injecting an appropriate `OBJECT` or `EMBED` element into an HTML document to checking if the right version of Silverlight is installed, and then directing the viewer to the appropriate place to install it if it isn't. You should always use the functionality in `Silverlight.js` (discussed in this section) rather than directly using `OBJECT` or `EMBED` unless your content must appear in an environment where JavaScript is not allowed.

## Silverlight.createObject

The simple function exposed by `Silverlight.js` is `Silverlight.createObject`. Here is how `createObject` could be called in JavaScript to generate an `OBJECT`/`EMBED` element as shown in Listings 1.1 and 1.3:

```
Silverlight.createObject(
  "Chapter1.xaml",                        // source XAML
  document.getElementById("placeholder"), // parent HTML element
  "silverlightControl",                   // id for the control
  // properties:
  { width: "390", height: "100", version: "1.0", background: "Yellow" },
  // events:
  {}
);
```

The first parameter becomes the `source` value for the dynamically generated `OBJECT` or `EMBED` element, and the third parameter becomes its `id`. The second parameter can be an existing HTML element to contain the new `OBJECT` or `EMBED` element. In this example, the standard `document.getElementById` function is used to retrieve an element from the page via its HTML `id` (`placeholder`), but you could also pass `document.body` if you want to append the new element directly to the page's body.

The fourth and fifth parameters to `createObject` are associative arrays of properties and events, respectively, supported by the Silverlight add-on. The properties array is a mix of values that

**TIP**

If you pass `null` for the parent HTML element, `createObject` returns a string containing the `OBJECT` or `EMBED` element that would have otherwise been added to the parent. This gives you some flexibility for morphing the element or otherwise customizing how it is added to your page.

either alter the logic inside Silverlight.js (such as version), are applied directly to the OBJECT or EMBED element (such as width and height), or are applied as PARAM element children when the OBJECT element is used (such as background). The various properties (and events) are covered in the upcoming "Understanding Your Hosting Options" section. The only new property shown here is version, which should simply be set to the version of Silverlight you're targeting (1.0).

**TIP**

The createObject function has sixth and seventh (optional) parameters that can both be used to attach custom data to the Silverlight control. For example, if you set the sixth parameter (initParams) to the string "custom", the dynamically generated OBJECT element would have the following additional child:

```
<param name="initParams" value="custom"/>
```

With this in place, you could write JavaScript that retrieves this value with standard DOM functions for traversing the tree of HTML elements or with a simple Silverlight-specific property called InitParams explained toward the end of this chapter. If you set the seventh parameter (context) to any object, that object will be passed as a parameter to the control's onLoad event handler (covered later in this chapter). This context functionality is specific to Silverlight.js and, unlike initParams, cannot be accomplished with a PARAM element in HTML.

The capabilities provided by these two mechanisms are simply additional ways to communicate information between JavaScript files that might be developed as separate components.

## Silverlight.createObjectEx

Silverlight.js defines a second function for embedding Silverlight content called Silverlight.createObjectEx. (The Ex suffix is an old Win32 convention that has mysteriously made its way into this file. It typically denotes a newer or "extra" version of a function.) The only difference between createObject and createObjectEx is that the latter accepts a single associative array parameter with all the same information. For example, here is the previous call to createObject translated into a call to createObjectEx:

```
Silverlight.createObjectEx(
  // Just one parameter, an array with 5 elements:
  {
    source: "Chapter1.xaml",
    parentElement: document.getElementById("placeholder"),
    id: "silverlightControl",
    properties:
      { width: "390", height: "100", version: "1.0", background: "Yellow" },
    events: {}
  }
);
```

The nice thing about `createObjectEx` is that calls to it are self-descriptive. You can clearly see what piece of data is the `source`, `parentElement`, and so on without the need for comments. For this reason, examples in this book use `createObjectEx` rather than `createObject`. The syntax for calling `createObjectEx` might look unusual, but it's basically JSON (JavaScript Object Notation), a popular data interchange format based on simple JavaScript constructs.

## DIGGING DEEPER

### The Implementation of `createObjectEx`

`createObjectEx` is a very simple wrapper over `createObject`, as you can see by looking at its source code inside `Silverlight.js`. It is effectively implemented as follows:

```
Silverlight.createObjectEx = function(params)
{
  return Silverlight.createObject(params.source, params.parentElement, params.id,
    params.properties, params.events, params.initParams, params.context);
}
```

In JavaScript, syntax such as `a.b` is equivalent to `a["b"]`, which is why `params.source` can be used to access the `source` element of the `params` array, and so on.

## Putting It All Together

The `createObject` or `createObjectEx` function can be called from any JavaScript file or inline `SCRIPT` element, but Microsoft has published the following recommended approach for using these functions:

1. Create a separate script file called `CreateSilverlight.js` (by convention).
2. Define a parameterless function (called `createSilverlight` by convention) inside `CreateSilverlight.js` that makes the call to `createObject` or `createObjectEx`.
3. Reference both `Silverlight.js` and `CreateSilverlight.js` from `SCRIPT` elements in your HTML document (usually inside the document's `HEAD`).
4. Place an HTML element that you want to contain the Silverlight content, such as a `DIV`, inside the document and choose an `id` (used by your `createSilverlight` function).
5. Call the parameterless function inside inline JavaScript in the HTML document.

## WARNING

### When calling `createObject` or `createObjectEx`, some properties and events can't be omitted!

If you omit the `version` property, you'll get a script error; and if you omit either the `width` or `height`, the resultant element won't be seen. As for `events`, you must at least specify an empty associative array (`{}`); otherwise, you'll get a script error.

Listings 1.4 and 1.5 follow this approach to get the same result pictured in Figures 1.1 and 1.3.

LISTING 1.4 Embedding Silverlight Content Using the Recommended Silverlight.js Approach

```
<html>
  <head>
    <title>Great Estates</title>
    <script type="text/javascript" src="Silverlight.js"></script>
    <script type="text/javascript" src="CreateSilverlight.js"></script>
  </head>
  <body style="background:blue">
    <!-- A Silverlight-based logo: -->
    <div id="placeholder">
      <script type="text/javascript">createSilverlight();</script>
    </div>
    <p style="font-family:Tahoma; color:white">
      An idyllic new community located high on a hill and offering captivating
      waterfront views. Tailored to meet both the needs of upsizing and
      downsizing buyers, Great Estates offers custom quality architecture and
      design at an affordable price point.
    </p>
  </body>
</html>
```

LISTING 1.5 CreateSilverlight.js—The Recommended Script File with the Parameterless createSilverlight Function

```
function createSilverlight()
{
  Silverlight.createObjectEx(
    {
      source: "Chapter1.xaml",
      parentElement: document.getElementById("placeholder"),
      id: "silverlightControl",
      properties:
        { width: "390", height: "100", version: "1.0", background: "Yellow" },
      events: {}
    }
  );
}
```

## DIGGING DEEPER

### Avoiding "Click to activate and use this control" in Internet Explorer

Depending on how ActiveX controls are used, current versions of Internet Explorer require viewers of a web page to "activate" it by clicking it (or pressing Enter or the spacebar when it has focus). Once activated, the control can accept keyboard and mouse input. Hovering over such controls shows a border and tooltip, as displayed in Figure 1.4.

This behavior is certainly annoying, but it is especially annoying for content that is supposed to blend seamlessly with HTML. For this example, why would a viewer of this page care about activating a logo? This anti-feature exists because of a recently settled patent case (Eolas v. Microsoft) that had required Microsoft to change Internet Explorer's handling of ActiveX controls.

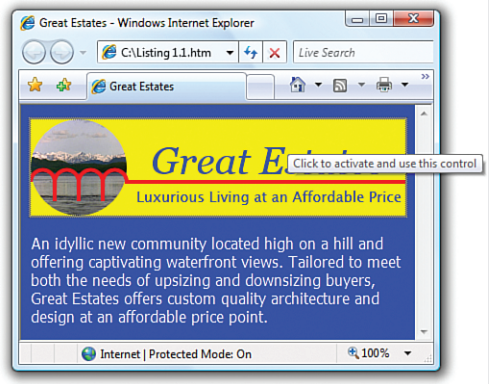


FIGURE 1.4 The annoying "Click to activate and use this control" behavior in Internet Explorer.

Fortunately, there are techniques for avoiding the activation behavior, as covered in various articles (such as http://msdn2.microsoft.com/en-us/library/ms537508.aspx). Even better, by following the recommended approach of using `Silverlight.js` and `CreateSilverlight.js`, you don't need to do anything further. This is why viewing the pages from Listings 1.1 and 1.3 gives the "Click to activate and use this control" prompt, but the page from Listing 1.4 (and the remaining examples in this book) does not.

## DIGGING DEEPER

### Silverlight Streaming by Windows Live

Silverlight Streaming by Windows Live is a web service that provides highly scalable hosting and streaming of Silverlight content free (with advertising) or for a small fee. This service has its own procedure to follow for packaging and uploading content, but the consumption of the content is very similar to the normal `Silverlight.js` approach. Instead of referencing your own copy of `Silverlight.js`, you can reference a modified `Silverlight.js` provided by Silverlight Streaming. Then you can call `Silverlight.createHostedObjectEx`—a special function defined by this service—which embeds an `IFRAME` into your HTML document rather than an `OBJECT` or `EMBED` element directly. The `source` given to `createHostedObjectEx` must be a special string containing pieces of information that you must previously register with the Silverlight Streaming service. Alternatively, you can leverage Silverlight Streaming without JavaScript by setting the source of an `IFRAME` to a special URL specific to your hosted application. For more details, go to http://streaming.live.com.

# Understanding Your Hosting Options



Silverlight exposes a number of properties and events that customize the appearance of the Silverlight content and the way it interacts with the HTML document it lives inside. In addition, the `source` parameter exposed by the Silverlight add-on supports more functionality than previously described. This section examines the extra functionality of `source`, and then looks at all the properties and events that the add-on directly exposes.

## `source`

Previous listings have demonstrated the most common usage of `source` setting it to the name (and path, if applicable) of a XAML file on the web server. However, you can alternatively place your XAML inline in the HTML document. There are two steps for doing this:

1. Place your XAML content within a `SCRIPT` element with type `text/xaml` somewhere in the document *before* the HTML element that will contain the Silverlight control, and give it a unique `id`.
2. Use the `SCRIPT` element's `id` preceded by a `#` as the `source` value given to the Silverlight add-on. The `#` prefix is what distinguishes an `id` from a filename.

Listings 1.6 and 1.7 are updates to Listings 1.4 and 1.5 that remove the dependency on the separate `Chapter1.xaml` file.

LISTING 1.6 Placing Inline XAML Inside HTML

```
<html>
  <head>
    <title>Great Estates</title>
    <script type="text/javascript" src="Silverlight.js"></script>
    <script type="text/javascript" src="CreateSilverlight.js"></script>
  </head>
  <body style="background:blue">
    <!-- A Silverlight-based logo: -->
    <script id="xaml" type="text/xaml">
      <Canvas xmlns="http://schemas.microsoft.com/client/2007">
        <MediaElement Name="video" Source="Lake.wmv" Opacity="0" IsMuted="true"/>
        <!-- A circle containing a live video: -->
        <Ellipse Width="100" Height="100">
          <Ellipse.Fill>
            <VideoBrush SourceName="video"/>
          </Ellipse.Fill>
        </Ellipse>
        <!-- Two pieces of text: -->
        <TextBlock FontFamily="Georgia" Foreground="Blue" FontStyle="Italic"
          FontSize="40" Canvas.Left="125" Canvas.Top="20" Text="Great Estates"/>
```

LISTING 1.6 Continued

```
        <TextBlock Foreground="Blue" Canvas.Left="110" Canvas.Top="70"
          Text="Luxurious Living at an Affordable Price"/>
        <!-- Curves and a line: -->
        <Path Stroke="Red" StrokeThickness="4">
          <Path.Data>
            <PathGeometry>
              <PathFigure StartPoint="0,65">
                <ArcSegment SweepDirection="Clockwise" Size="2,2" Point="25,65"/>
                <ArcSegment SweepDirection="Clockwise" Size="2,2" Point="50,65"/>
                <ArcSegment SweepDirection="Clockwise" Size="2,2" Point="75,65"/>
                <ArcSegment SweepDirection="Clockwise" Size="2,2" Point="100,65"/>
                <LineSegment Point="390,65"/>
              </PathFigure>
            </PathGeometry>
          </Path.Data>
        </Path>
      </Canvas>
    </script>
    <div id="placeholder">
      <script type="text/javascript">createSilverlight();</script>
    </div>
    <p style="font-family:Tahoma; color:white">
      An idyllic new community located high on a hill and offering captivating
      waterfront views. Tailored to meet both the needs of upsizing and
      downsizing buyers, Great Estates offers custom quality architecture and
      design at an affordable price point.
    </p>
  </body>
</html>
```

LISTING 1.7 `CreateSilverlight.js`—Using Inline XAML as the source

```
function createSilverlight()
{
  Silverlight.createObjectEx(
    {
      source: "#xaml",
      parentElement: document.getElementById("placeholder"),
      id: "silverlightControl",
      properties:
        { width: "390", height: "100", version: "1.0", background: "Yellow" },
      events: {}
    }
  );
}
```

This #id syntax is supported anywhere the source might be specified: createObject, createObjectEx, directly on an EMBED element, or as a PARAM inside an OBJECT element. This functionality is a handy way to combine what would ordinarily be two web requests into one. But in addition to efficiency considerations, removing the dependency on a custom external file enables server-side code (in technologies such as ASP.NET or PHP) to emit Silverlight content in a completely encapsulated way.

## Properties

The width, height, and version properties exposed by Silverlight are straightforward, but the background property could use a little more explanation. In addition, the Silverlight add-on supports more properties that haven't been discussed yet.

### background

The background property—which can be set via createObject, createObjectEx, or directly on an OBJECT/EMBED element—is more powerful than a normal HTML color value. Besides named colors—such as Red or Yellow—and RGB values—such as #F1F1F1 or #456, background can be given an alpha channel for creating transparent or translucent background colors. The syntax is #AARRGGBB (or #ARGB), so a translucent red color would be #77FF0000 (or #7F00). background can also be set to the named value Transparent, which is the same as any color with an alpha channel value of zero. If you omit background altogether, the control will be given a white background.

### isWindowless

By default, an instance of the Silverlight control is known as *windowed*, but by setting isWindowless to true (which can be done via createObject, createObjectEx, or directly on an OBJECT/EMBED element), you can change it to be a *windowless* control. The distinction of windowed versus windowless isn't specific to Silverlight, but rather refers to a low-level implementation detail on Windows (whether the control has its own window handle, or HWND).

**WARNING**

**Inline XAML doesn't work in Firefox unless the DOCTYPE element is removed!**

Putting a DOCTYPE (document type declaration) in your HTML page that specifies which version of HTML or XHTML you're using is a best practice. However, current versions of Firefox have a bug that prevents inline XAML from working on a page with a DOCTYPE. Therefore, if you care about your content rendering on Firefox, you must choose to use only one or the other.

**WARNING**

**The XAML file used as the source must be served from the same domain as the web page!**

You cannot set the Silverlight control's source to a different domain (or protocol) than the one hosting the HTML document. This limitation is intentional, as a security measure. Although this restriction is unnecessarily strict (in this author's opinion), it is at least consistent with the policy that browsers enforce with their XmlHttpRequest object, called the *same origin policy*. (People have come to believe that XML from a different domain is inherently more dangerous than JavaScript from a different domain, because all browsers block the former but allow the latter! I wouldn't be surprised to see browsers change their policy in the next few years.)

**TIP**

In addition to using literal strings, you can set `background` to the color of any existing HTML element. For example, the following call gives the Silverlight control a background color that matches the host document, if it has one set via the `style` attribute:

```
Silverlight.createObjectEx
  {
    ...
    properties:
      { ... , background: document.body.style.backgroundColor },
    ...
  );
```

This is much preferred to using a background color of `Transparent`, because it works regardless of other Silverlight property settings and it can give dramatically better performance.

The important thing to understand is the two different behaviors of a windowless control:

- A windowless control respects HTML z-indexing, so you can overlay and overlap HTML content on top of Silverlight and vice versa. A windowed control, on the other hand, is always rendered on top.
- A windowless control supports transparency, so it can be given a transparent or translucent background, and content inside it can be transparent or translucent.

Figure 1.5 shows a potential way that the Great Estates website might take advantage of windowless Silverlight content—placing an HTML `SELECT` element on top of the Silverlight logo.

To create the result in Figure 1.5, Listing 1.8 adds a `SELECT` element to the page from Listing 1.4 and uses CSS to give it an absolute position and a z-index to ensure that it is placed on top of the Silverlight content.

**WARNING**

**Transparent or translucent `background` colors only work as expected if `isWindowless` is set to `true`!**

Without setting this to `true`, a `background` set to `Transparent` will appear black, and translucent colors will be blended with black rather than the HTML content behind the Silverlight control.

**WARNING**

**Using a windowless control or a transparent/translucent background can severely degrade performance!**

The performance problems with windowless controls and colors with an alpha channel are especially apparent in Safari on Mac OS X. Therefore, unless the behavior enabled by windowless controls and transparent/translucent content is absolutely necessary, you should avoid using these features.

**TIP**

Despite the performance implications, many rich Internet applications created with Silverlight 1.0 need to set `isWindowless` to `true`. The ability to place HTML-based controls (whether simple controls similar to `INPUT` or `BUTTON` or richer controls such as those found in ASP.NET AJAX) on top of Silverlight content is crucial, due to the lack of such controls natively existing in Silverlight. With a windowless control, you can even overlay Flash on top of Silverlight content! Microsoft Popfly is an example of a rich Internet application that does all these things. If you can confine your Silverlight content and HTML content to regions that don't overlap, however, then you can get away with a windowed control.

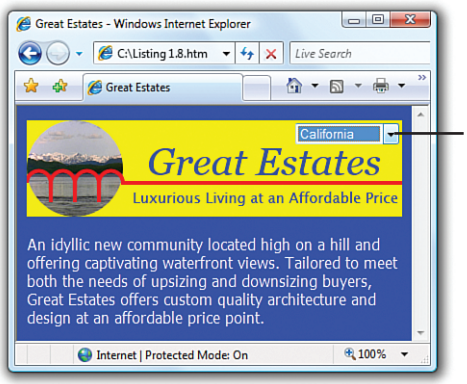


FIGURE 1.5 A windowless Silverlight control allows HTML to appear on top of it.

LISTING 1.8 Placing an HTML `SELECT` Element in Front of the Silverlight Control

```
<html>
  <head>
    <title>Great Estates</title>
    <script type="text/javascript" src="Silverlight.js"></script>
    <script type="text/javascript" src="CreateSilverlight.js"></script>
  </head>
  <body style="background:blue">
    <!-- A Silverlight-based logo: -->
    <div id="placeholder">
      <script type="text/javascript">createSilverlight();</script>
    </div>
    <select style="position:absolute; left:289px; top:18px; z-index:1">
      <option>California</option>
      <option>Pennsylvania</option>
      <option>Washington</option>
    </select>
    <p style="font-family:Tahoma; color:white">
      An idyllic new community located high on a hill and offering captivating
```

LISTING 1.8 Continued

```
      waterfront views. Tailored to meet both the needs of upsizing and
      downsizing buyers, Great Estates offers custom quality architecture and
      design at an affordable price point.
    </p>
  </body>
</html>
```

Listing 1.8 only produces the desired result because the corresponding `CreateSilverlight.js` file sets `isWindowless` to `true`, as shown in Listing 1.9.

LISTING 1.9 `CreateSilverlight.js`—Hosting Familiar Silverlight Content in a Windowless Control

```
function createSilverlight()
{
  Silverlight.createObjectEx(
    {
      source: "Chapter1.xaml",
      parentElement: document.getElementById("placeholder"),
      id: "silverlightControl",
      properties:
        { width: "390", height: "100", version: "1.0", background: "Yellow",
          isWindowless: "true" },
      events: {}
    }
  );
}
```

**WARNING**

**The Boolean used for `isWindowless` must be specified as a string!**

The following property setting works in a call to `createObject` or `createObjectEx`:

```
{ ... , isWindowless: "true", ... }
```

But the following setting does not work as expected:

```
{ ... , isWindowless: true, ... }
```

Any non-string is treated as `false`, and therefore has no effect!

### `inplaceInstallPrompt`

The `inplaceInstallPrompt` property, which can only be used with `createObject` or `createObjectEx`, controls the look and behavior of the Silverlight installation graphic that gets displayed when the viewer doesn't have the appropriate version of Silverlight.

Figure 1.6 shows the appearance of the two options. Setting `inplaceInstallPrompt` to `false` (the default behavior) gives a small graphic that links to the official download page with more information. Setting it to `true` gives additional text, but the link now points directly to the file to download rather than an intermediate page.


FIGURE 1.6 The two different install prompts supported by `Silverlight.js`.

**WARNING**

**The Boolean used for `inplaceInstallPrompt` must *not* be specified as a string!**

Unlike the case for `isWindowless`, the following property setting works in a call to `createObject` or `createObjectEx`:

```
{ ... , inplaceInstallPrompt: false, ... }
```

But the following setting does not work as expected:

```
{ ... , inplaceInstallPrompt: "false", ... }
```

Any string is treated as `true`!

### `maxFramerate`

The `maxFramerate` parameter, which can be set via `createObject`, `createObjectEx`, or directly on an `OBJECT/EMBED` element, customizes the maximum frame rate that the Silverlight control renders content, measured in frames per second. (The actual frame rate is dependent on the client computer and its current load.) The default value for `maxFramerate` is 24. If you decide to customize `maxFramerate`, you should select the lowest number possible that gives you the results you need.

The frame rate controls all content inside the Silverlight control—animations and even video—except for audio. You can see this with the Great Estates logo by setting its `maxFramerate` to 1 and changing `IsMuted` to `false` instead of `true` in the XAML file. This causes the video to progress in an extremely choppy fashion, yet the corresponding audio plays smoothly.

## WARNING

**The number used for `maxFramerate` must be specified as a string!**

Similar to `isWindowless`, the following property setting works in a call to `createObject` or `createObjectEx`:

```
{ ... , maxFramerate: "24", ... }
```

But the following setting does not work as most people would expect:

```
{ ... , maxFramerate: 24, ... }
```

Any non-string is treated as zero frames per second!

## DIGGING DEEPER

**`maxFramerate` Versus `framerate`**

You might come across some Silverlight examples that set the `framerate` property instead of `maxFramerate`. Setting `framerate` is exactly the same as setting `maxFramerate`, and it can only be done via `createObject` or `createObjectEx`. The logic in `Silverlight.js` maps both `framerate` and `maxFramerate` to the one true `maxFramerate` property supported by the underlying Silverlight control. It does this simply for compatibility with prerelease versions of Silverlight. For clarity, you should stick to using `maxFramerate` if you feel the need to customize the frame rate.

## LOOKING FORWARD

**The `enableHtmlAccess` Property**

Silverlight also supports a property called `enableHtmlAccess`, but it only applies to versions after 1.0. It controls whether .NET code (such as C#) is capable of accessing the browser's DOM via a special layer designed for .NET. The default value of `enableHtmlAccess` is `true`, but it doesn't apply to JavaScript hosted by the browser because it always has access to the browser's DOM.

## Events

The Silverlight control supports two events that can be set directly on the `OBJECT` or `EMBED` element: `onLoad` and `onError`. You can assign either event the name of a JavaScript function to be called. For example:

```
<object type="application/x-silverlight" id="silverlightControl"
  width="390" height="100">
  <param name="background" value="Yellow"/>
  <param name="source" value="Chapter1.xaml"/>
  <param name="onLoad" value="myFunction"/>
</object>
```

However, because handling either of these events requires the use of JavaScript, you might as well take advantage of `createObject` or `createObjectEx` rather than attaching these handlers the "raw" way.

### onLoad

The `onLoad` event is raised as soon as the XAML content has been loaded. Handling this event is useful for performing custom initialization of Silverlight content, such as initiating animations or dynamic positioning/sizing of the control based on document dimensions. These specific kinds of activities are covered in later chapters, but Listing 1.10 at least demonstrates how to designate a function as a handler for the `onLoad` event.

LISTING 1.10 `CreateSilverlight.js`—Assigning an `onLoad` Handler

```
function createSilverlight()
{
  Silverlight.createObjectEx(
    {
      source: "Chapter1.xaml",
      parentElement: document.getElementById("placeholder"),
      id: "silverlightControl",
      properties:
        { width: "390", height: "100", version: "1.0", background: "Yellow" },
      events: { onLoad: myFunction }
    }
  );
}
function myFunction(control, context, rootElement)
{
  // Perform custom initialization
}
```

The purpose of Silverlight's `onLoad` event is similar to the HTML DOM's `onload` event. However, to avoid timing issues, you should stick to the HTML `onload` event for manipulating HTML content and Silverlight's `onLoad` event for manipulating Silverlight content.

Handlers for the `onLoad` event are passed three parameters:

- **`control`**, which is the instance of the Silverlight control. The next section, "Interacting with the Silverlight Control Programmatically," describes some of the things you can do with this object.
- **`context`**, which is simply whatever custom `context` value was given to `createObject` or `createObjectEx` (if one was given).
- **`rootElement`**, which is the instance of the root element in the `source` XAML content. The next chapter explains how you can programmatically interact with Silverlight elements declared in XAML.

**WARNING**

**The function for `onLoad` (and `onError`) must not be specified as a string!**

Unlike the strings passed as most property values, the elements in the `events` associative array must contain direct references to the functions you've defined (function pointers), as in Listing 1.10. The following would cause a script error:

```
{ onLoad: "myFunction", ... }
```

### `onError`

The `onError` event is raised whenever Silverlight throws an exception not already handled by your JavaScript code. (For an exception thrown from a synchronous function call, this means that no corresponding `try/catch` block exists. For an exception thrown from an asynchronous function call, this means that no event handler is attached for that specific failure case.) Exceptions can be raised by Silverlight for XAML parsing errors or for any number of runtime errors.

If you don't specify a handler for `onError` when directly using an `OBJECT` or `EMBED` element, unhandled Silverlight errors are swallowed. But when you use `createObject` or `createObjectEx`, a function called `default_error_handler` is automatically set as the handler for `onError` unless you provide your own. The default handler calls JavaScript's `alert` function to display a simple dialog, such as the one shown in Figure 1.7.

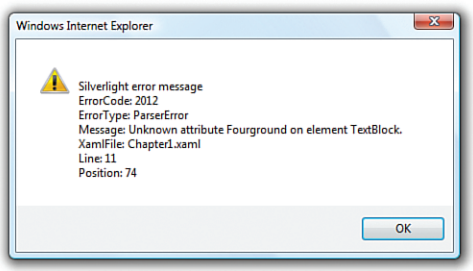


FIGURE 1.7 When good content goes bad.

To understand how to create your own `onError` handler, it is instructive to look at the implementation of `default_error_handler` inside `Silverlight.js`. It is effectively implemented as follows:

```
function default_error_handler(sender, args)
{
  var errMsg = "\nSilverlight error message\n";
  // All errors have a numeric code, a type, and a message
  errMsg += "ErrorCode: " + args.errorCode + "\n";
  errMsg += "ErrorType: " + args.errorType + "\n";
  errMsg += "Message: " + args.errorMessage + "\n";
  if (args.errorType == "ParserError")
  {
    // A parser error gives the location in the XAML content
    errMsg += "XamlFile: " + args.xamlFile + "\n";
    errMsg += "Line: " + args.lineNumber + "\n";
    errMsg += "Position: " + args.charPosition + "\n";
  }
```

```
  else if (args.errorType == "RuntimeError")
  {
    if (args.lineNumber != 0)
    {
      // Display the line number and character, if the information exists
      errMsg += "Line: " + args.lineNumber + "\n";
      errMsg += "Position: " +  args.charPosition + "\n";
    }
    // The name of the function that failed
    errMsg += "MethodName: " + args.methodName + "\n";
  }
  // Display the message in a simple alert box:
  alert(errMsg);
}
```

The `sender` is the object on which the error occurred, if applicable. For parser errors, such as the one shown in Figure 1.6, `sender` is `null`. The `args` object provides a number of pieces of information that depend on the type of error raised, as seen in the implementation of `default_error_handler`.

**TIP**

Despite the presence of an `onError` handler, it's easy to make a mistake in JavaScript causing an error that doesn't get sent to this function. The behavior of such unhandled JavaScript errors varies from browser to browser. To debug them in Internet Explorer with Visual Studio, be sure to uncheck the **Disable script debugging** settings in the **Advanced** tab of the **Internet Options** pane!

## Interacting with the Silverlight Control Programmatically

The `OBJECT` or `EMBED` element representing the Silverlight control (whether part of the static HTML document or dynamically injected by `Silverlight.js`) has an HTML `id`, so you can write JavaScript to retrieve the element and get or set properties on it just like any other HTML element. For example,

```
// Retrieve the element via a standard HTML DOM function:
var element = document.getElementById("silverlightControl");
// Set properties on the element:
element.width = 500;
element.style.zIndex = 2;
```

Because this element is an instance of the ActiveX object (or Netscape plug-in), it provides a number of useful properties, functions, and events specific to Silverlight. This element returned by `document.getElementById` is the same object passed as the first parameter to the `onLoad` event handler. However, you should avoid accessing any Silverlight-specific members on this object before the control has finished loading (and its `onLoad` event is raised).

The Silverlight control exposes most of its functionality via two properties: `Settings` and `Content`.

## The `Settings` Property

Most relevant to this chapter is the control's `Settings` property, which defines a number of subproperties for getting or setting a number of attributes (many of which could have alternatively been set via `createObject`, `createObjectEx`, or directly on the `OBJECT/EMBED` element):

- **`Background`**—The same property discussed earlier. However, this makes it easy to change the background color at any time.
- **`EnableFramerateCounter`**—A Boolean property that toggles the display of the current frame rate in the browser's status bar. (This is potentially useful for debugging purposes.)
- **`EnableRedrawRegions`**—Another Boolean property meant for debugging, this highlights regions of the screen that are redrawn on each frame, when set to `true`.
- **`EnableHtmlAccess`**—The same property discussed earlier.
- **`MaxFrameRate`**—The same property discussed earlier.
- **`Windowless`**—The same as the `isWindowless` property discussed earlier.

For example, the `EnableRedrawRegions` and `Background` properties can be set in a Silverlight `onLoad` event handler as follows:

```
function myFunction(control, context, rootElement)
{
  control.Settings.EnableRedrawRegions = true;
  control.Settings.Background = "Red";
}
```

These properties, and all other members exposed on the control object, are pretty flexible. For example, they are not case sensitive. Many people prefer using lowercase names because it matches JavaScript conventions, as in the following code that produces the same result as the preceding snippet:

```
function myFunction(control, context, rootElement)
{
  control.settings.enableRedrawRegions = true;
  control.settings.background = "Red";
}
```

In addition, the Boolean properties can be set to a `true` or `false` string *or* to a `true` or `false` Boolean literal, and they work correctly either way.

None of the `Settings` members are extremely compelling, however, as it's rare you would need to retrieve or change the data after the control has loaded.

## The `Content` Property

The most commonly used member on the Silverlight control is its `Content` property, which represents the XAML content hosted by the control and exposes some interesting functionality. It has the following subproperties:

- **`ActualWidth` and `ActualHeight`**—Report the dimensions of the Silverlight control. You can discover the same information by using the HTML DOM, although these Silverlight properties give different values than the corresponding HTML properties when the browser zoom level (an Internet Explorer feature) is not 100%. These Silverlight properties always report the real dimensions, whereas the HTML properties report the virtual dimensions (in essence, hiding the zoom level).
- **`Root`**—The instance of the root element in the current XAML content. This is the same object passed to `onLoad` as the `rootElement` parameter. (This property makes the `rootElement` parameter unnecessary because the handler can always use `control.Content.Root` instead.)
- **`FullScreen`**—Enables the Silverlight content to fill the entire screen. To prevent hostile Silverlight applications from holding your screen hostage, full-screen mode must be initiated by a user action (such as a mouse click or key press). Therefore, this functionality is covered in Chapter 7, "Responding to Input Events."
- **`Accessibility`**—Enables you to customize how the Silverlight control appears to accessibility software. The `Accessibility` object contains three settable properties: `Title`, `Description`, and `ActionDescription` (see Chapter 7 for more information).

`Content` exposes three functions explained in Chapter 2, "XAML," and Chapter 8, "Downloading Content on Demand":

- **`CreateFromXaml`**—Dynamically creates Silverlight content specified in XAML in a JavaScript string.
- **`CreateFromXamlDownloader`**—Dynamically creates Silverlight content specified in a XAML file downloaded on demand.
- **`FindName`**—Finds the instance of a Silverlight object defined in XAML based on an assigned name.

`Content` even exposes two unique events that cannot be consumed any other way. For example, you cannot specify either of these in the `events` array passed to `createObject` and `createObjectEx`. These two events are

- **`OnResize`**—Raised whenever the value of `Content`'s `ActualWidth` or `ActualHeight` property changes
- **`OnFullScreenChange`**—Raised whenever the value of `Content`'s `FullScreen` property changes

A handler can be attached to either event by assigning a function reference. Listing 1.11 demonstrates this for the `OnResize` event.

LISTING 1.11 `CreateSilverlight.js`—Assigning an `OnResize` Handler

```
function createSilverlight()
{
  Silverlight.createObjectEx(
    {
      source: "Chapter1.xaml",
      parentElement: document.getElementById("placeholder"),
      id: "silverlightControl",
      properties:
        { width: "390", height: "100", version: "1.0", background: "Yellow" },
      events: { onLoad: myFunction }
    }
  );
}
function myFunction(control, context, rootElement)
{
  control.Content.OnResize = function()
  {
    var htmlElement = document.getElementById("silverlightControl");
    alert("Actual Dimensions: " + control.Content.ActualWidth + "x" +
                                  control.Content.ActualHeight);
    alert("Virtual Dimensions: " + htmlElement.offsetWidth + "x" +
                                   htmlElement.offsetHeight);
  };
}
```

In this example, `OnResize` is set to a JavaScript closure (a function defined inside another function), which displays the control's dimensions according to Silverlight and according to the HTML DOM. If you try this with any of the examples in this chapter and change Internet Explorer's zoom level to 200%, you'll see that the HTML DOM still reports dimensions of 390x100 but Silverlight reports dimensions of 780x200. Although Internet Explorer doesn't want web pages to know when they are being zoomed (because they could do weird things that interfere with proper zooming), leveraging this information can be critical for Silverlight content because the visual elements inside the control do not get scaled automatically. Chapter 6, "Positioning and Transforming Elements," discusses the resizing of Silverlight content.

## Other Members

In addition to the `Settings` and `Content` properties, the Silverlight control defines three more properties:

- **InitParams**—Gives whatever string was set (if any) for the `initParams` parameter to `createObject` or `createObjectEx` (or directly on the `OBJECT/EMBED` element). Although `InitParams` is always exposed to JavaScript as a single string, a comma-delimited list will be split into an array of strings passed to .NET code in future versions of Silverlight.
- **IsLoaded**—Reports whether the Silverlight content has been loaded.
- **Source**—Gives the control's `source` URL or `#id` value. This property can also be set to a new URL or `#id` value. This causes the control to reload with the new content, and the `onLoad` event will be raised again.

The control also directly defines two functions:

- **CreateObject**—Enables you to create an instance of the downloader object described in Chapter 8.
- **IsVersionSupported**—Given an input string containing a version number such as `1.0`, this function tells you whether the installed version of Silverlight is compatible with that version. `Silverlight.js` uses this internally to perform its version checking.

The control also defines a single event—**OnError**—that is the same as the `onError` event described earlier. By assigning a function reference to the control's `OnError` member, you can change the default error handler at any time. Note that the control does not have an `OnLoad` member. You can only assign a handler for the `onLoad` event using the approaches discussed earlier.

## Conclusion

As time passes, more software is targeted for the Web, and more software is expected to deliver high-quality—sometimes *cinematic*—experiences. However, the effort involved in creating such user interfaces has been far too difficult in the past.

If you're a software developer, you might be skeptical about the need for "eye candy" beyond what HTML provides. But like it or not, having an engaging user experience matters, whether you are creating a public consumer-facing site, or a simple intranet application for your manager. You can blame the unrealistic software on movies and on TV, or you can blame real-world software that is starting to catch up to Hollywood's standards! Indeed, modern software has more visual polish than it used to. You can see it in traditional operating systems (such as Mac OS X and, more recently, Windows Vista), in software for devices such as TiVo or Xbox, and of course all over the Web thanks to Adobe Flash. Users have increasing expectations for the experience of using software, and companies are spending a great deal of time and money on user interfaces that differentiate themselves from the competition. Microsoft understands this, and it's apparent in its latest technologies—first on the desktop with WPF, and now on the Web with Silverlight.

Silverlight makes it easier than ever before to create engaging web-based user interfaces, whether you want to create a simple piece of content or an immersive interactive experience

worthy of a role in a summer blockbuster! This chapter focused on the HTML and/or JavaScript required for getting any Silverlight content inside a web page, as well as the ways in which the embedding can be customized. The next chapter explores the XAML side of the story in depth, and then the remainder of the book covers all the different types of content and interactivity that can be achieved with Silverlight.