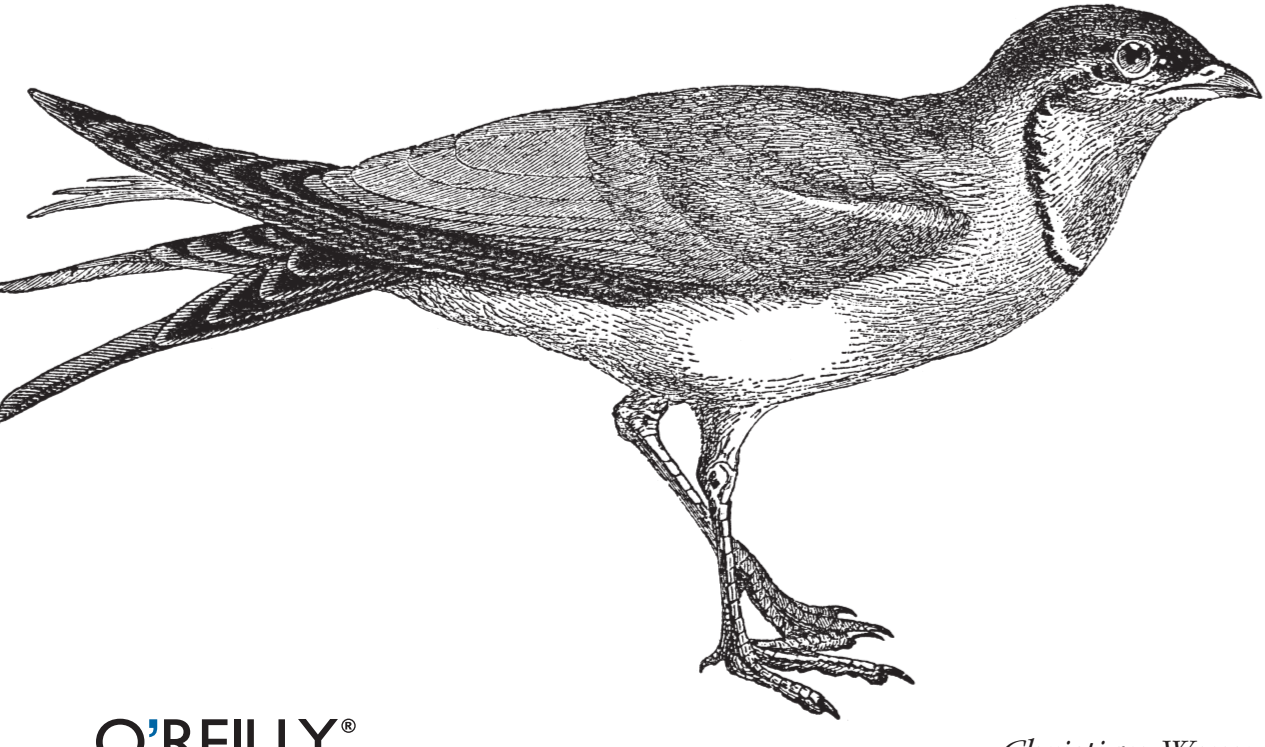
Get started now with
Microsoft's new cross-platform plug-in
for rich internet applications

Covers 1.0 with 1.1 Preview

Essential

Silverlight



O'REILLY®

Christian Wenz

Essential Silverlight

Essential Silverlight

Christian Wenz

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

Essential Silverlight
by Christian Wenz

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472

Editor: John Osborn
Copy Editor: Laurel R.T. Ruma
Production Editor: Laurel R.T. Ruma

Cover Designer: Karen Montgomery
Interior Designer: David Futato

Printing History:
September 2007: First Edition.

ISBN-10: 0-596-51611-8
ISBN-13: 978-0-596-51611-6

Table of Contents

Preface

I would describe myself as a web guy. When I first accessed the World Wide Web sometime around 1994, I immediately fell in love with the possibilities and technical challenges. From then on, I almost exclusively worked on web projects and did very little programming apart from that. In all those years since then, I have seen technologies come and go, but some of them stayed. For instance, I remember starting to work with ASP and PHP simultaneously in about 1997 or 1998, and finally moving away from ASP because it was so limited. I returned to the ASP world when the first betas of ASP.NET were released, and my interest heightened when ASP.NET 2.0 came up, and it was off to the races again. (Today, I am happily using both.) I appreciate that my JavaScript knowledge is in demand again, thanks to one new term: Ajax.

One of the technologies I really developed a love/hate relationship with was Macromedia Flash (now Adobe Flash). I really like that the technology can do so much more than HTML and JavaScript, including everything you want to call "Ajax." I am also happy that the browser plugin has such an enormous market share. I really, really hate the Flash editor. The designers I work with are very happy with it, but from a developer's perspective, I change into explicit lyrics mode whenever I have to use it. This is probably no surprise: Flash is historically a designer's tool and has just recently begun to appeal to developers. I am a terrible designer, so probably I do not deserve better.

But still, Flash is a very nice technology, since it combines advanced graphical features with powerful coding support. So I was more than happy when I heard that Microsoft was working on a similar technology: Silverlight. (No one at Microsoft will ever tell you that there is a connection between Silverlight and Flash, and that's probably true, but it serves to point out similarities and differences.) Knowing that Microsoft has always been a very developer-friendly company, I expected the features of Flash, with a better development experience (at least for me). And, to be honest, the first steps are really promising. Most programming is done in trusted Visual Studio, and there are designer tools as well. Microsoft has still a long way to go, both on the tool itself and also with regards to the market share, but the first steps are done, and I am looking forward to seeing the next steps.

Who This Book Is For

There are two target audiences for this book: developers who would like to familiarize themselves with the Silverlight technology, and designers who would like to see what Silverlight has to offer. My focus, however, is on the developer's side. This book does not try to provide a complete reference to Silverlight. It is true to the concept of the *Essentials* series: you will get Silverlight up and running soon, see the most important concepts, and will find lots of code examples.

There are currently two Silverlight versions available, 1.0 (released on September 4, 2007), and 1.1 (currently a alpha version). This book covers Silverlight 1.0, and only provides a short preview to the upcoming version (which will come out sometime in 2008). Knowledge of Windows Presentation Foundation (WPF) is not required, but if you have already worked with it, you may already know some Silverlight basics. From a programming point of view, JavaScript is the language of choice. If you have not worked with that language before, refer to the O'Reilly catalog for some excellent choices.

How This Book Is Organized

Part 1 contains background information on Silverlight and related technologies.

Chapter 1
: Introduces Windows Presentation Foundation (WPF) and how it relates to Silverlight.

Chapter 2
: Goes through all required installation steps and creates your first Silverlight application.

Chapter 3
: Reviews software tools that facilitate creating Silverlight content.

Part 2 focuses on the results you can achieve with the declarative means of Silverlight; but some JavaScript coding will also be covered.

Chapter 4
: Features the most important elements of Microsoft's WPF markup language.

Chapter 5
: Explains how Silverlight applications may become interactive by processing events.

Chapter 6
: Exposes two different approaches to making Silverlight animations dynamic.

Chapter 7
: Shows how to use audio and video data in Silverlight applications, including JavaScript access.

Part 3 focuses on development aspects.

Chapter 8
: Describes how to access Silverlight content from JavaScript.

Chapter 9
: Shows advanced JavaScript possibilities, including the ability to make HTTP requests.

Chapter 10
: Reviews how two technologies—ASP.NET AJAX and Silverlight—combine their powers.

Chapter 11
: Looks at the upcoming Silverlight version 1.1.

Appendix A
: Provides a list of properties and methods the Silverlight plugin exposes.

What You Need to Use This Book

For developing Silverlight content, you only need a text editor. It is much more convenient is to use Visual Studio 2005 or the (free) Visual Web Developer Express Edition 2005. Chapter 3 covers these and additional tools. Chapter 2 guides you through all necessary installation steps both for developing and for viewing Silverlight content.

Conventions Used in This Book

The following typographical conventions are used in this book:

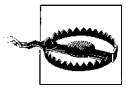
Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This icon signifies a tip, suggestion, or general note.



This icon indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Essential Silverlight* by Christian Wenz. Copyright 2007 O'Reilly Media, Inc., 978-0-596-516-116."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

When you see a Safari® Books Online icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at *http://safari.oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707 829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/9780596516116

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

http://www.oreilly.com

Acknowledgments

I have expressed on various occasions that technical book authors should not thank their partners/kids/dogs and pretend that writing a book put their private and social life at risk. When reading prefaces of fiction or memoirs you never hear such complaints, but in technical books they seem to be all too common. I have written several dozens of those and always managed to juggle work and play.

This time, however, I understood. This book was written on an extremely tough schedule so that it could be published in time with the Silverlight release. So, I had to work crazy hours and neglect some things and some people. (Not that I haven't done that in the past, but this time it was worse than usual.) Therefore, thanks to all who suffered in one way or another, you know who you are.

I also have to thank my editor at O'Reilly, John Osborn, for joining forces with me again. Andrew Savikas got me set up with DocBook and also tried to convince me that writing a book in XML is not too bad (I still want my word processor back). Keith Fahlgren set up the Subversion repository, implemented the automated PDF build, and also cleaned up my DocBook mess from time to time. Laurel Ruma copy edited the text, and I don't know what she cursed more: my writing or my XML. Yvonne Schimmer provided me with video material for the chapter on multimedia and supported the rest of the book as well.

Finally, I do have to thank my excellent technical reviewers: WPF guru Rouven Haban and vector graphics and Flash expert Tobias Hauser. Thank you for your hard work, and should you find any errors left, I introduced them intentionally right before the book was sent to the printer.

PART I

Introduction

CHAPTER 1

WPF Basics

Of Vectors and Pixels

Most graphics nowadays are pixel-based. Every point in the graphic is represented by one pixel. This is a really good solution for most scenarios, including digital photography (where you really want to maintain every single information the camera is "seeing"), but there are shortcomings too. For instance, have a look at Figure 1-1, where you see a simple text created in Microsoft Paint. This text is pixel based. In Figure 1-2, you see the same text, but this time the image width and height have been enlarged. Do you see the stairway effect? So when you make a pixel-based image larger, you lose quality.

That's obvious, of course. Imagine, for example, that an 100x100 pixel image is resized to 200x200 pixels. Instead of 10,000 pixels, we how have 40,000 pixels. So, where we had a 1x1 pixel in the original image, we now have 2x2 pixels. Paint is using a very simple algorithm to resize images: if the graphic becomes larger, just clone the pixels. This then creates the stairway effect.

Professional software like Adobe Photoshop comes with several quite sophisticated algorithms to make the quality loss when resizing images less severe (especially when making them smaller); however, there still is a notable effect when increasing an image's dimensions.

There is an alternative approach: vector-based images. Every element on an image is a geometrical object: a line, a circle, a polygon, a curve, just to name a few. The main advantage is that there is no quality loss when resizing the image: a circle just changes its width, but that's all. There is no stairway effect, since it is still a circle and not, as with a pixel image, a set of pixel ordered in a circular fashion.

Obviously, not every image can be represented as vectors. Think again of photos–it is theoretically possible to try to find geometrical elements and patterns in a portrait or in a landscape (there even are algorithms for that!), but it is virtually impossible to create an exact representation of a photo by just using vectors. However, in computing there

Figure 1-1. A text in Microsoft Paint

are several areas where vectors make real sense. One such area is fonts. See Figure 1-3 for a typical Windows font (coincidentally the same font used in Figure 1-1 and Figure 1-2). Most fonts are vectors, so there is no quality difference whether you use them in 8pt, 10pt, 12pt, or 100pt. If you type a letter in a word processor and then change the font size to something really high, you still get smooth edges. Once you paste a text into a pixel-based imaging software like Paint, you lose the vector information and are back to pixels.

Another area prone for vector use is user interface (UI). The standardized UIs are, most of the time, vector-compatible. Most of them need to be resizeable, so the content should remain intact if the user chooses the width of the height of a window. However, in reality, very few UIs have been really based on vectors.

WPF

Some time ago the folks at Microsoft sat down and designed the next generation of UI technology for their Vista operating system (names were different back then, of course). The system should be vector-based and use XML. The final system is called Windows Presentation Foundation (WPF).

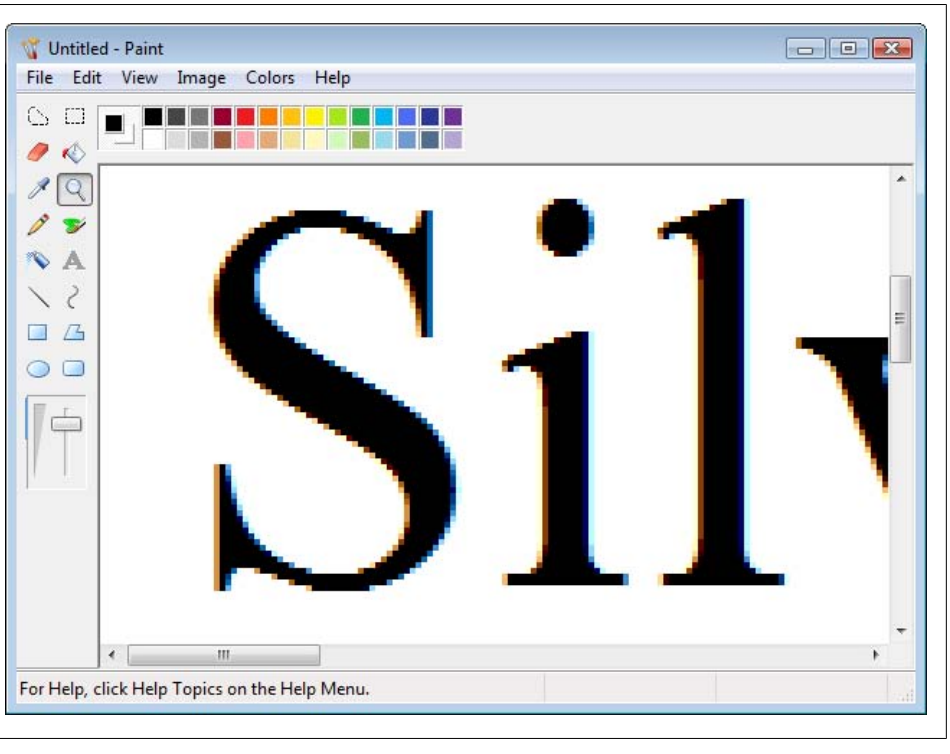


Figure 1-2. The same text, increased in size

Vector Graphics and XML

There were (and still are) several other projects trying to create vector graphics (and maybe some animation or business logic support) using XML. One of the oldest is the World Wide Web Consortium's (W3C) Scalable Vector Graphics (SVG). SVG graphics are created using XML, support scripting, and are supported by most modern browsers except Internet Explorer. However, SVG has not reached mainstream market penetration yet, so it is only successful in some niche markets, including mobile phones and cartography.

Another related approach comes from Adobe. Flex uses yet another XML dialect (called MXML) to dynamically create Flash content.

WPF is an integral part of the .NET Framework 3.0, which is installed by default on Windows Vista and is an additional download for Windows XP and Windows 2003. Other acronyms that are part of .NET 3.0 include:

WCF (Windows Communication Foundation)
: Communication subsystem

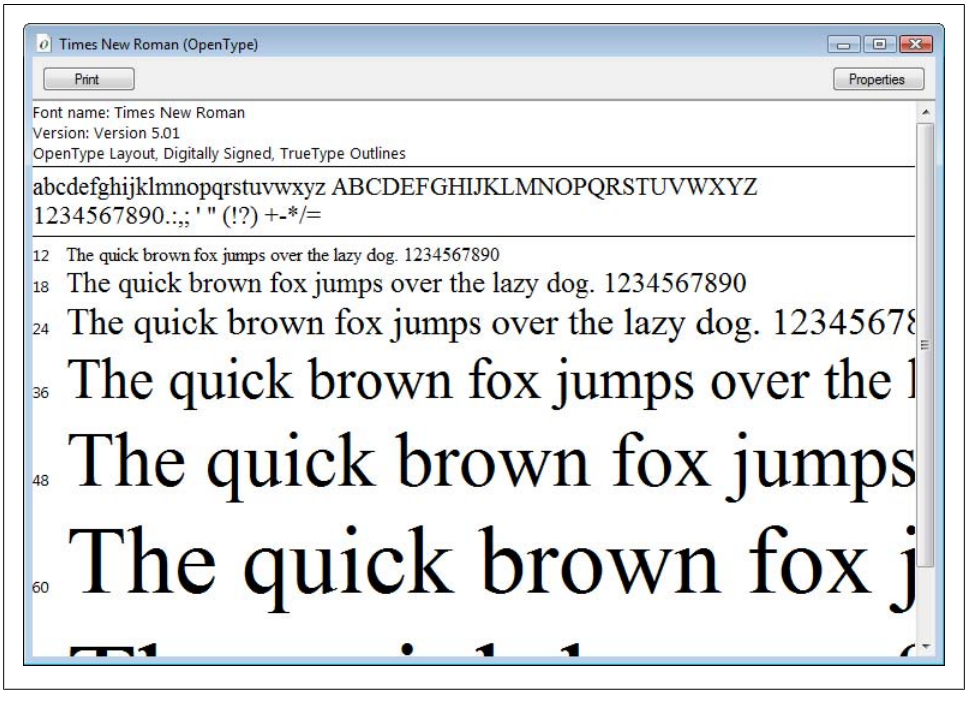


Figure 1-3. A vector-based Windows font

WCS (Windows CardSpace)
: Digital identity subsystem

WF (Windows Workflow Foundation)
: Workflow subsystem

WPF applications either run in the browser or as standalone desktop applications. Both scenarios require that .NET Framework 3.0 or higher is available. Currently, most WPF applications are standalone because Vista does not have a high market penetration and .NET 3.0 is quite a hefty download. Therefore, Microsoft created a similar technology targeted at the browser world: Silverlight. Let's first look at WPF to get some more background information.

The main focus of WPF lies on vector graphics, but pixelated graphics are supported as well. WPF also supports multimedia content in forms of audio and video data. One of the highlight features is the support for text, which includes some typographical specialities like justified text, kerning, and ligatures.

It is certainly no surprise that all coding (in terms of business logic) is done using .NET languages like C# and Visual Basic. The .NET Framework—or to be exact, the CLR (Common Language Runtime)—defines every possible type of element in a WPF application and enables a good development experience in Visual Studio (think Intelli-

Figure 1-4. The Forbes news reader uses WPF

Sense) and rapid prototyping. Also, the API access to XAML offers more than XAML itself, so in order to get the most out of XAML/Silverlight, you need to familiarize yourself with both markup and code.

There are already several WPF prototype applications, including several "virtual newspapers" that showcase text flow, such as the Forbes.com reader (see Figure 1-4; download it at *http://www.forbes.com/partners/microsoft/newsreader/*). The next version of the Yahoo! messenger will also feature a slick WPF interface.

XAML

But didn't the previous section just mention that the WPF content is created using XML? Indeed, there is a special format (or XML dialect) for that purpose: XAML (eXtensible Application Markup Language). It is used for the UI markup in WPF applications. The WPF runtime then interprets this markup, displays the UI and also integrates the additional business logic code (which is, as aforementioned, written in a .NET language like C# or VB).

Microsoft also provides several tools to develop XAML content. You can use Visual Studio, but for a more visual experience, Expression Blend (part of the Microsoft Expression Suite) is an interesting option. The .NET Framework 3.0 SDK also contains

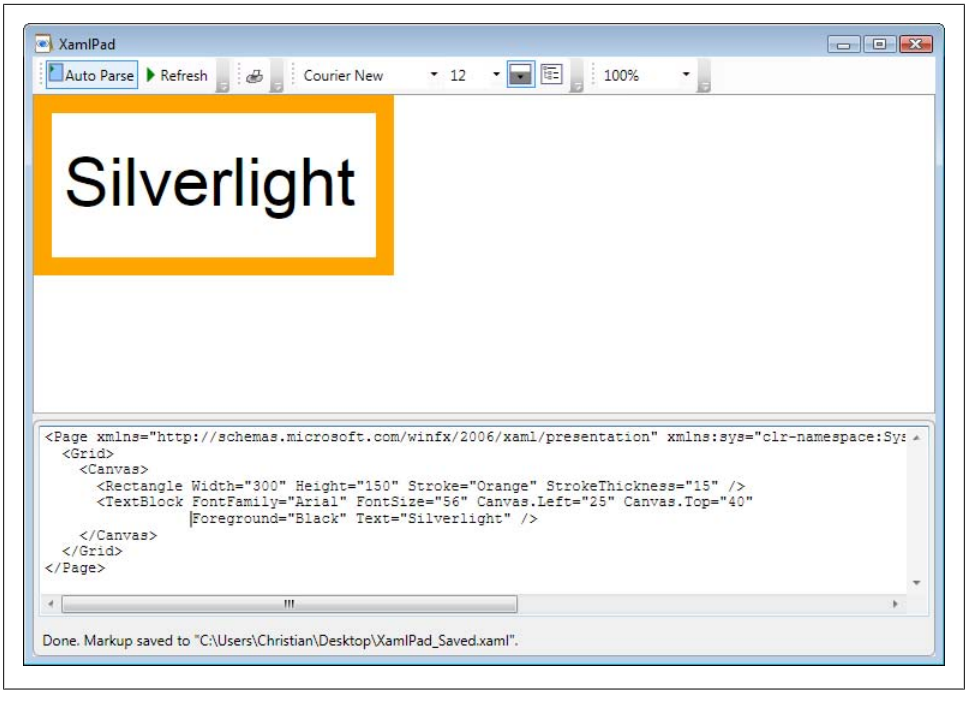


Figure 1-5. XAMLPad shows both XAML and the visual output

an application called XAMLPad that features a split view: You see both XAML markup and the actual WYSIWYG appearance of the code at the same time. Figure 1-5 shows XAMLPad in action.

When creating Silverlight content, you don't have to worry about WPF because the .NET Framework 3.0 is not required to develop or view Silverlight content. However, you should familiarize yourself a bit with XAML, since Silverlight supports a subset of XAML to create the UI. Therefore, Chapter 4 will introduce you to the most important XAML elements that are supported by Silverlight.

Further Reading

Programming WPF (http://www.oreilly.com/catalog/9780596510374/index.html) by Chris Sells and Ian Griffith (O'Reilly)

CHAPTER 2

Getting Started With Silverlight

About Silverlight

Some people refer to Microsoft's Silverlight technology as a "Flash killer," but I'm not sure whether that is really true. However, the similarities are striking. Both Adobe Flash (formerly Macromedia Flash) and Silverlight come as browser plugins. Both support vector graphics, audio and video playback, animations, and scripting support.

The technology basis is different. Flash uses a semi-open binary format, Silverlight is based on WPF. Before it was called Silverlight, the technology was codenamed WPF/E (Windows Presentation Foundation Everywhere). And thanks to good browser support, Silverlight can really be run everywhere, at least in theory.

In practice, the penetration of the browser plugin is a key issue. At the time of this writing, Silverlight plugins are available for the Windows platform (no surprise here) and support the two big players, Microsoft Internet Explorer and Mozilla Firefox. Also, a Mac OS X plugin exists that targets Safari and Mozilla Firefox on the Apple platform. According to Microsoft, other platforms were investigated, but given that Windows has such a high market share in terms of desktop operating systems and Mac OS X is number two on that list, these browsers were given priority.

Silverlight needs Windows XP or higher to run; the 1.0 version might, at some point, be updated to target Windows 2000. Opera support is also planned to be part of a future release; it is currently the third most used browser and has a relatively small but very loyal user base.

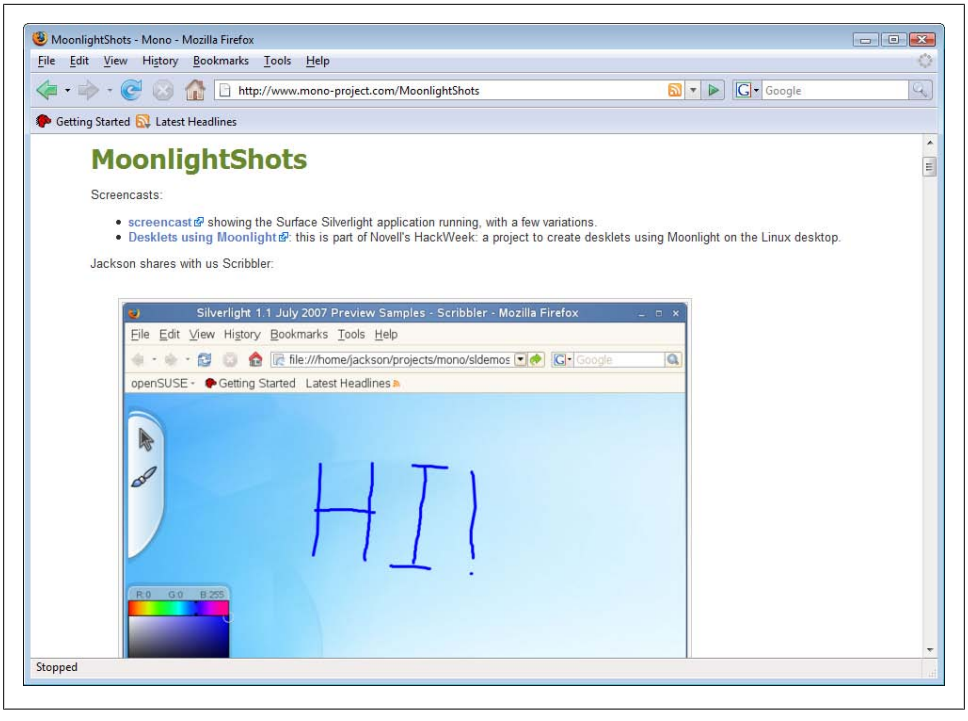


Figure 2-1. Some Moonlight screenshots

The Mono project (*http://www.mono-project.com/*), renowned for its open source implementation of .NET, is working on an open source Silverlight port that targets the Linux platform (and the Firefox browser there). This project is called Moonlight (see *http://www.mono-project.com/Moonlight* for a sneak peek). *http://www.mono-project.com/MoonlightShots* also shows some up-to-date screenshots of those efforts (see Figure 2-1). Microsoft has also announced that it is teaming with Novell to support the Moonlight project and bring Silverlight to Linux.

But what exactly is Silverlight, apart from a browser plugin provider? The heart of the plugin is the graphics subsystem, which supports a certain subset of WPF (see Chapter 4 for details). It also includes the codes responsible for displaying audio and video content (see Chapter 7 for more information on including multimedia content).

The architecture of Silverlight is quite complex (see *http://msdn2.microsoft.com/en-us/library/bb404713.aspx* for an overview), but it can be broken down into big chunks. The presentation system takes care of everything UI, including animation, text rendering, and audio/video playback. The plugin itself integrates into the browser so that the content can be shown, as well as accessed using the JavaScript DOM. Finally, using some JavaScript code (or, optimally, the ASP.NET AJAX framework), Silverlight applications can be enriched to access server APIs like web services. Figure 2-2 shows the

architecture. Silverlight 1.1 will further extend this and offer a partial .NET Framework integration right into Silverlight.

Setting Up a Silverlight Development System

For the programming part of Silverlight, a text editor would suffice, actually, but it is by far more productive if you use a "real" development environment. The most obvious choice is to use some of Microsoft's offerings. From a code perspective, Visual Studio 2005 is currently the best choice for developing Silverlight 1.0 content. Both the full versions (Standard Edition, Professional Edition, or Team Suite) and the free Visual Web Developer Express Edition work. If you can use a paid version, you will get project template support, so that's preferable. We will use Visual Studio 2005 Standard Edition throughout this book. Whenever there are differences to the free Express Edition, this will be especially noted so those users don't miss out on any important information.

There are no special prerequisites for installing Visual Studio or Visual Web Developer apart from using Windows XP or higher. You do not even need a web server, as the IDE comes with one! However, if you can, you may want to install Microsoft's IIS (Internet Information Services). They are hidden in the control panel, under *Software* (Vista: *Programs*), where you can turn Windows features on and off (see Figure 2-2).

When installing Visual Studio 2005, make sure that you select the *Visual Web Developer* option (see Figure 2-3). Otherwise, the web editor will not be part of your IDE, which you need to create web sites since Silverlight is a web technology, whereas WPF is a desktop technology. If you want to use Visual Studio 2005 Express Edition, you can download a web-based installer at *http://msdn.microsoft.com/vstudio/express/vwd/download/*.

Whatever version of Visual Studio you install, you should apply any available service packs, right away. (As time of writing this, Service Pack 1 was the most current one.) Windows Vista users must install a special Vista update patch as well. During installation, you'll get a notice that there is a known problem with running the software on Vista, and the solution is to install all available service packs and patches (see Figure 2-4).

Probably the most convenient way to get up-to-date regarding software patches for Microsoft products is the built-in Windows Update mechanism. Microsoft Update is an extended version of that service. Windows Update only gives you patches for Windows and core Windows components like Internet Explorer, but Microsoft Update also patches other Microsoft products, including Office, Visual Studio, and SQL Server (see Figure 2-5).

Activating Microsoft Update depends on which Windows edition you are using. For versions prior to Windows Vista, just go to the update web site (*http://update.microsoft.com/microsoftupdate/*), which will install the feature. If you are using Windows Vista, launch Windows Update from the start menu and then choose the

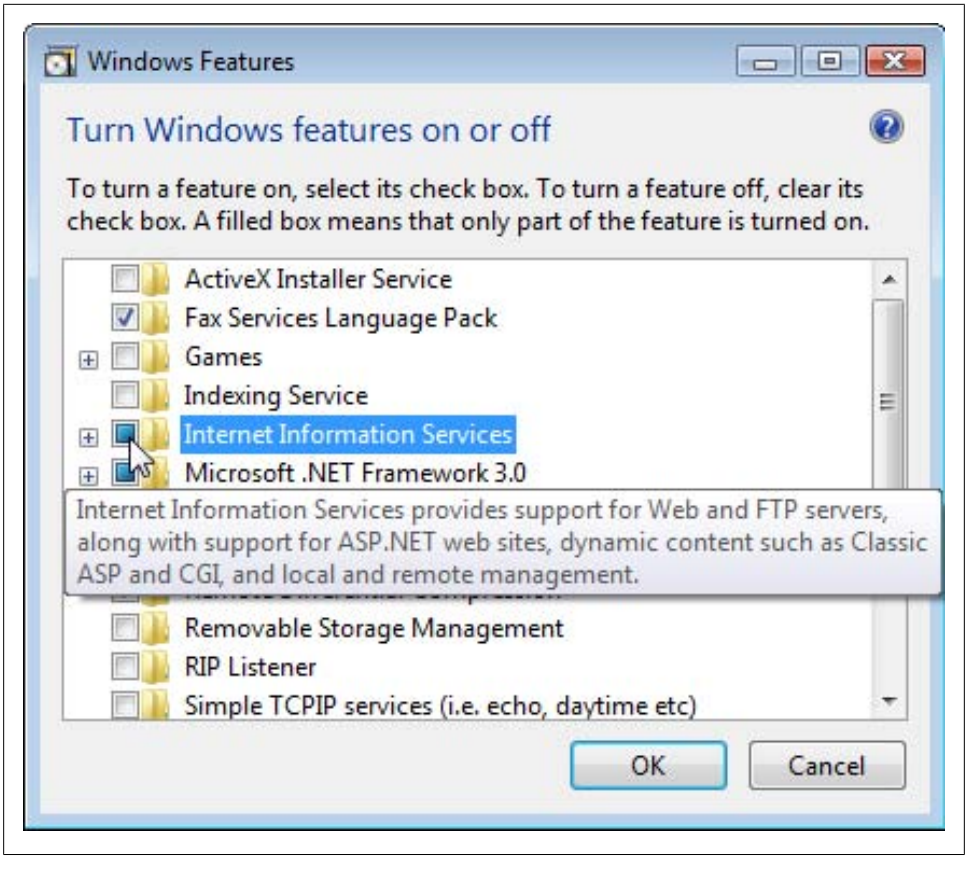


Figure 2-2. Installing IIS

Get updates for more products link (see Figure 2-6). The next time you search for updates, you will also get patches for Visual Studio (and other installed Microsoft software).

After installing Visual Studio 2005 Service Pack 1, Vista users need to run Microsoft Update again, to get a special Visual Studio update for their operating system.

For Windows Vista users, this is unfortunately not the end of the work to get Visual Studio running. When you launch the software (of course only *after* installing the updates), you will get the notice from that warns you that you need administrator privileges to have access to all Visual Studio features (see Figure 2-7). So if you can, right-click on the start menu shortcut to Visual Studio and select *Run as Administrator* (see Figure 2-8). If your system does not allow that or if you do

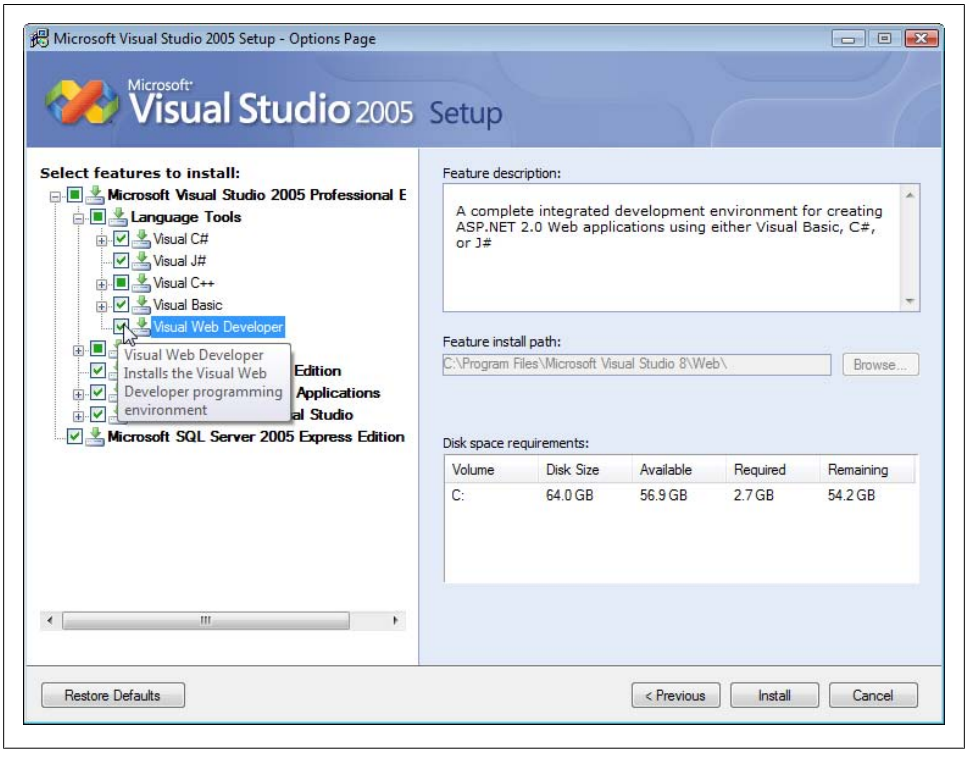
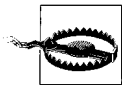


Figure 2-3. Make sure you install Visual Web Developer as part of Visual Studio 2005

not want to run software with full privileges, Visual Studio will still work, however some features (including debugging) will not work.

Once the IDE is up and running, it is time to make it Silverlight-aware. For both Silverlight 1.0 and 1.1, Microsoft is providing SDKs. We are using the 1.0 version here (see Figure 2-9). The final version of SIlverlight 1.0 SDK is available in the Microsoft download center at *http://www.microsoft.com/downloads/details.aspx?FamilyId=C72F125F-A6F6-4F4E-A11D-6942C9BA1834&displaylang=en*. It installs both samples and documentation, and also offers to install a Visual Studio 2005 template. If you accept that (which you really should do!), Visual Studio gets a new C# web site project template section for Silverlight (see Figure 2-10). Starting with a template like this really facilitates all subsequent steps, since a web site based on these templates comes with a lot of helper code so you don't have to type it all.



You need Visual Studio 2005 to install the templates. However, Visual Web Developer cannot use them, but in Chapter 10 you will find another convenient way that saves you quite some typing while setting up a Silverlight page.

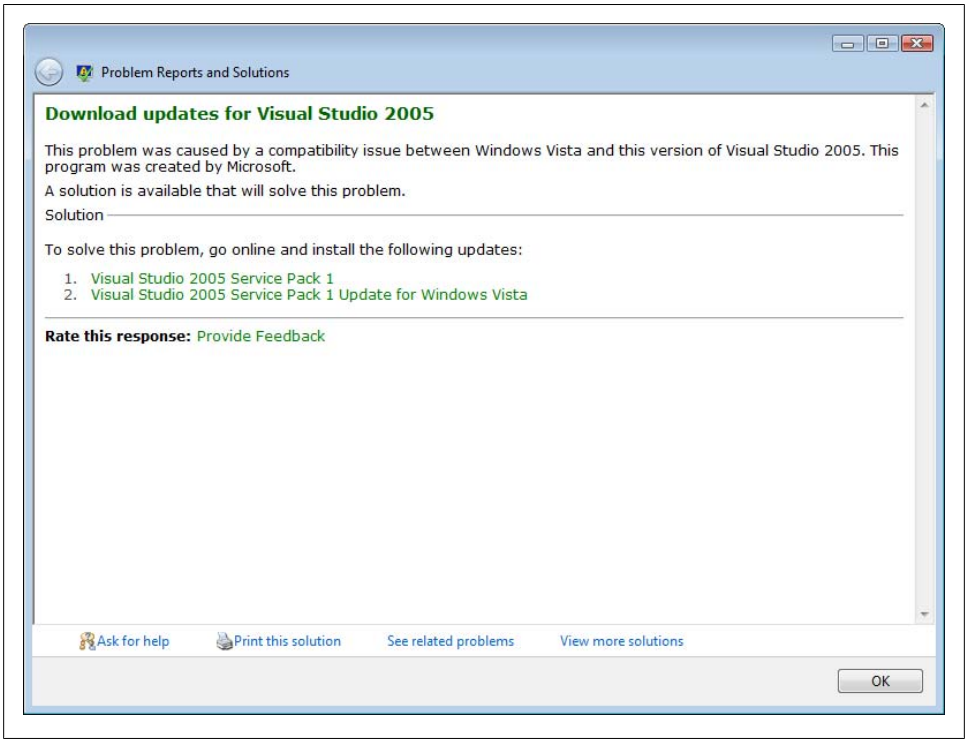


Figure 2-4. Running Visual Studio 2005 on Windows Vista requires some extra work

Now you are ready to create Silverlight content, at least in a code editor. For some other, more WYSIWYG choices, refer to Chapter 3.

If you are already using Visual Studio 2008 and want Silverlight JavaScript IntelliSense, the CodePlex project (*http://www.codeplex.com/intellisense*) provides with that functionality.

A First Silverlight Example

First of all, we need to setup a Silverlight project. Thanks to the Silverlight SDK's Visual Studio template, this is a relatively easy step. If you are using Visual Web Developer Express Edition, you don't have the luxury to use a project, so you need to create all files manually. Probably the best way is to download the book's samples from *http://www.oreilly.com/catalog/9780596516116* and start off with the files there.

In Visual Studio, choose *File/New Project* (not *File/New Web Site*!), and open up the *Visual C#* node (see Figure 2-10). There, you will find a *Silverlight Javascript Application* entry (ignore that *JavaScript* is not capped correctly).

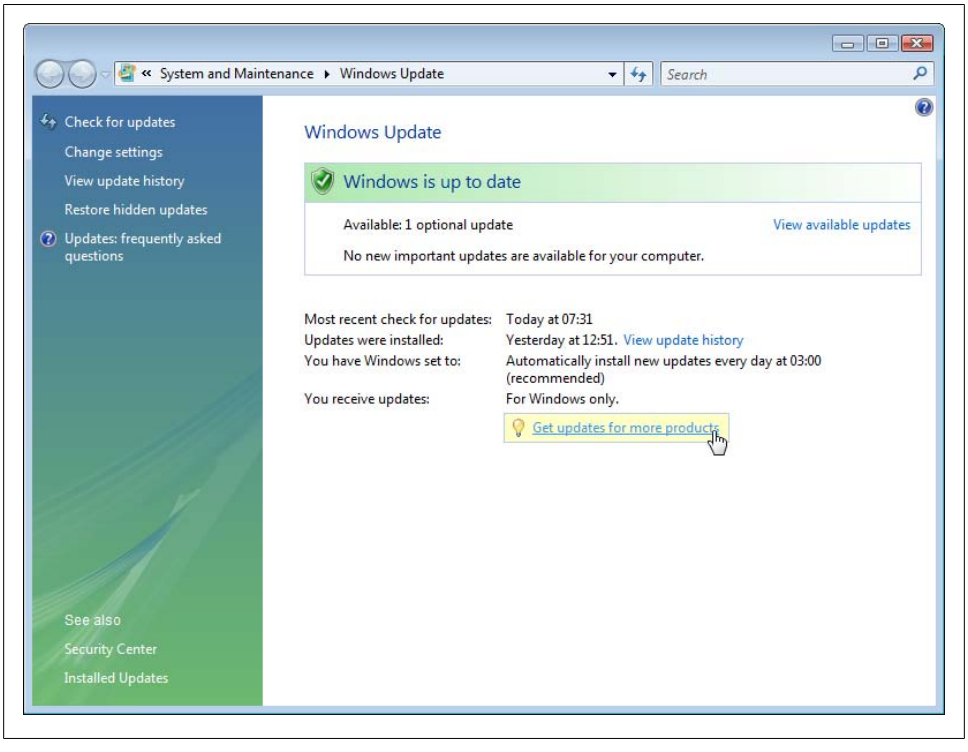


Figure 2-5. Switch to Microsoft Update to get more than just Windows updates

Throughout this book, the application that will be created here is the basis for all examples. We chose to call it `Silverlight`, you can of course also choose another name. By default, the project uses the built-in development of Visual Studio and assigns a random port. This port will be 12345 throughout the book, but all examples of course also work on other free ports and also when using the IIS instead.

The web site that the Silverlight template creates initially consists of the following five files:

Default.html
: An HTML page that contains markup to load Silverlight content

Default.html.js
: JavaScript code that loads Silverlight content

Silverlight.js
: A JavaScript helper library that is used by the *Default.html.js* file

Scene.xaml
: A sample XAML file

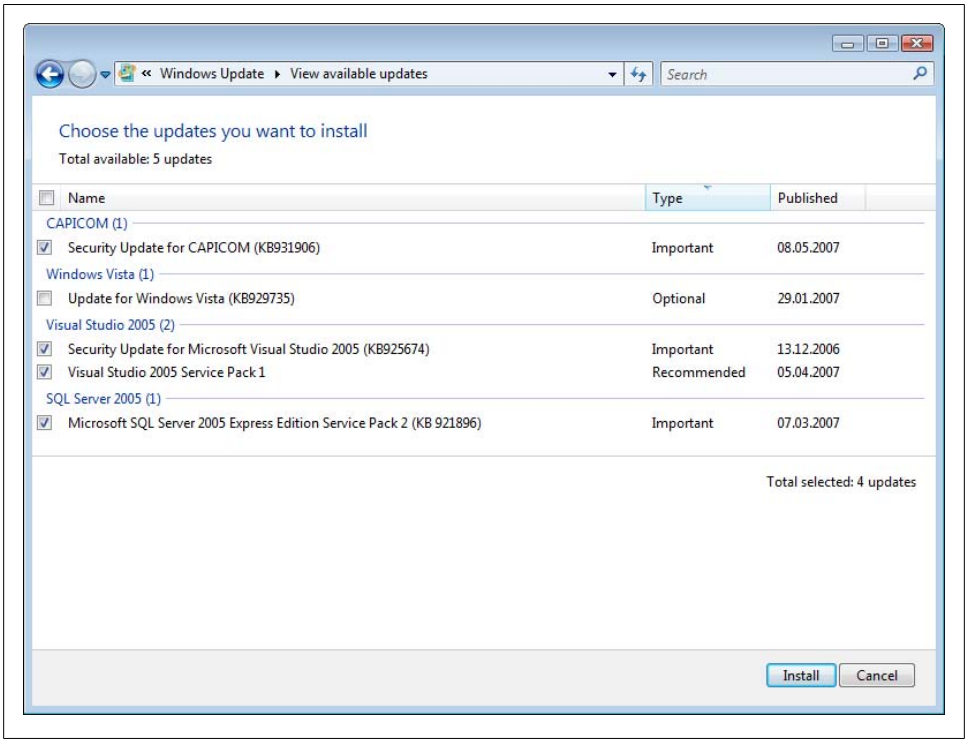
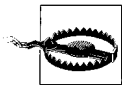


Figure 2-6. Microsoft Update offers you more updates, including those for Visual Studio

Scene.xaml.js
: JavaScript "code-behind" file for the XAML sample



On one of my systems, I kept getting strange error messages stating that Visual Studio could not access the *Default.html.js* file. I later found out that the guilty party was my antivirus software. By default, Windows does not show file extensions, so *Default.html.js* shows as *Default.html*. Because a JavaScript file may contain malicious code (especially if run locally), some viruses use this technique and my antivirus software wanted to protect me from that danger. All I could do at that point was move my Silverlight development into a secured environment and disable the resident shield of my antivirus software.

First of all, open the *Default.html* file and run the solutions (F5 for debugging mode, Ctrl-F5 for release mode). A browser window will open, but instead of fancy Silverlight content, you will get a message stating that Silverlight needs to be installed. (If you have already installed Silverlight, you will directly see the content, of course.) Figure 2-11 shows how this looks, regardless of what supported browser and operating system you are using.

Figure 2-7. Visual Studio prefers Administrator privileges

The plugin comes as an installation program; Figure 2-12 shows the Windows version. You may need to restart your browser afterward. After installing Silverlight, the content will appear, as Figure 2-13 shows.

Before you dive deeper into the world of Silverlight, let's have a closer look at the files that came with the template. We will *not* look at the XAML file (and the associated JavaScript script), because it contains quite a number of different techniques that will all be covered throughout this book. Let's indeed start with the *Default.html* file. It is reprinted in Example 2-1.

Example 2-1. The sample HTML file (Default.html.js)

```
<!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>Silverlight</title>

  <script type="text/javascript" src="Silverlight.js"></script>
  <script type="text/javascript" src="Default.html.js"></script>
  <script type="text/javascript" src="Scene.xaml.js"></script>
</head>

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

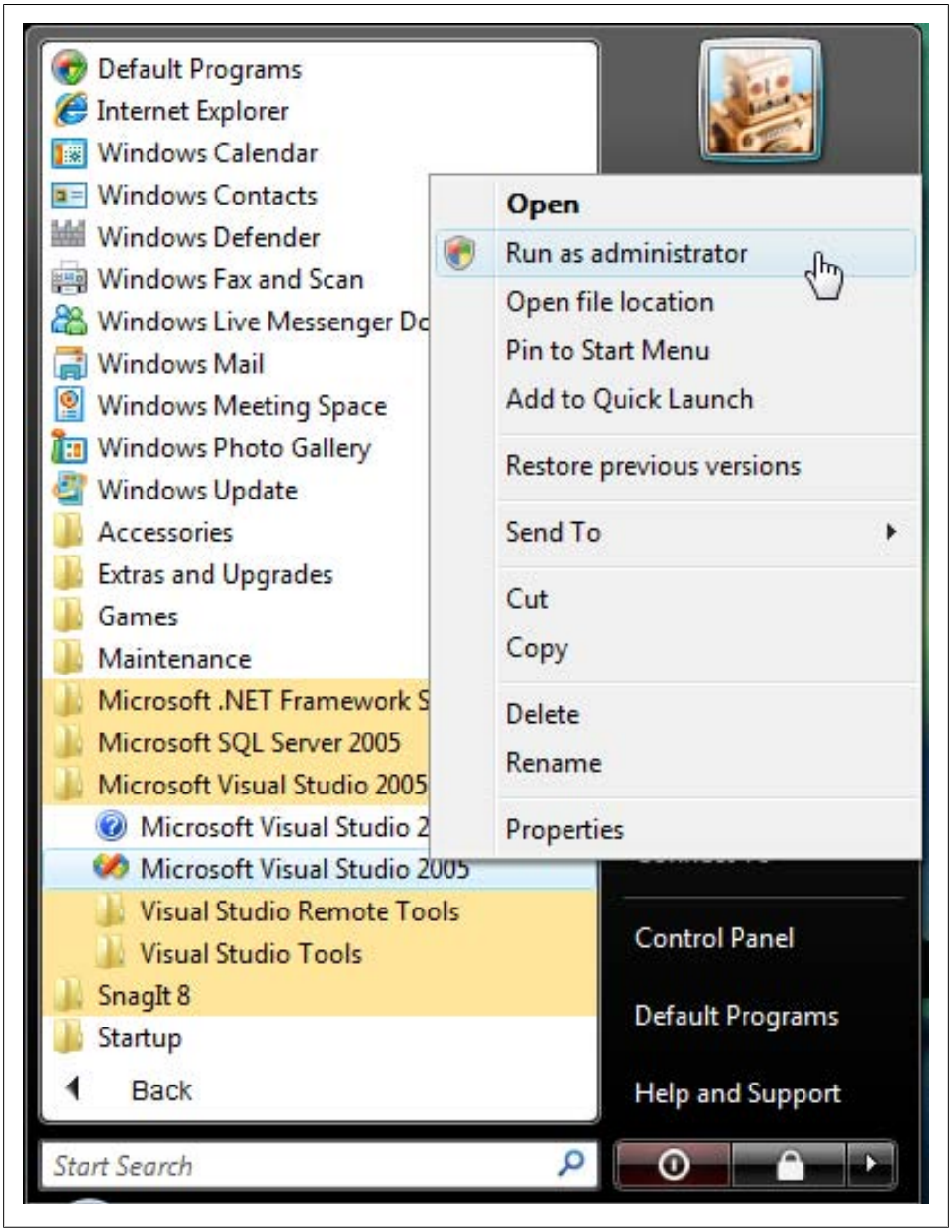


Figure 2-8. Run Visual Studio as an Administrator

```
    </script>
</div>
```

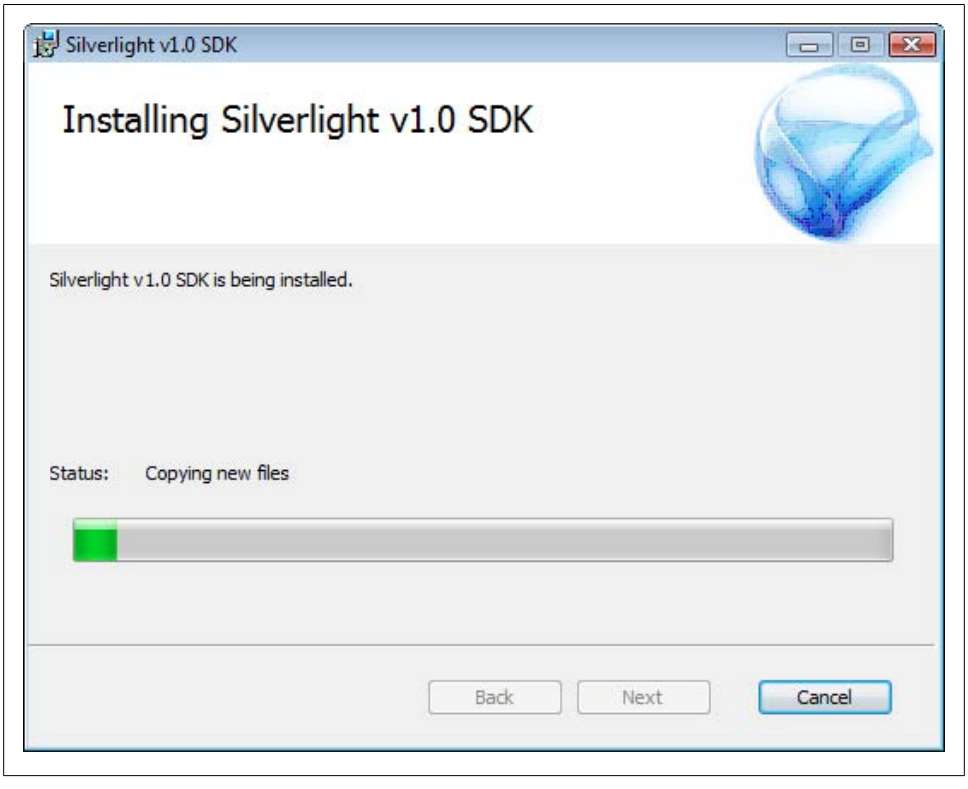


Figure 2-9. The Silverlight SDK Installer

```
</body>
</html>
```

So quite a number of things happen in this file:

- The *Silverlight.js* helper library is loaded with a `<script>` element.
- The *Default.html.js* JavaScript file (the "code-behind" script of *Default.html*) is loaded with a `<script>` element.
- The *Scene.xaml.js* JavaScript file (the "code-behind" script of the sample XAML file) is loaded with a `<script>` element.
- The page contains a `<div>` element with ID `"SilverlightPlugInHost"`, which will later hold the Silverlight content.
- The JavaScript function `createSilverlight()` (which, by the way, is defined in *Default.html.js*) is executed.

You especially need to remember the ID of the `<div>` container because it will later hold the Silverlight file. However, you need tell JavaScript (and the Silverlight) plugin ex-

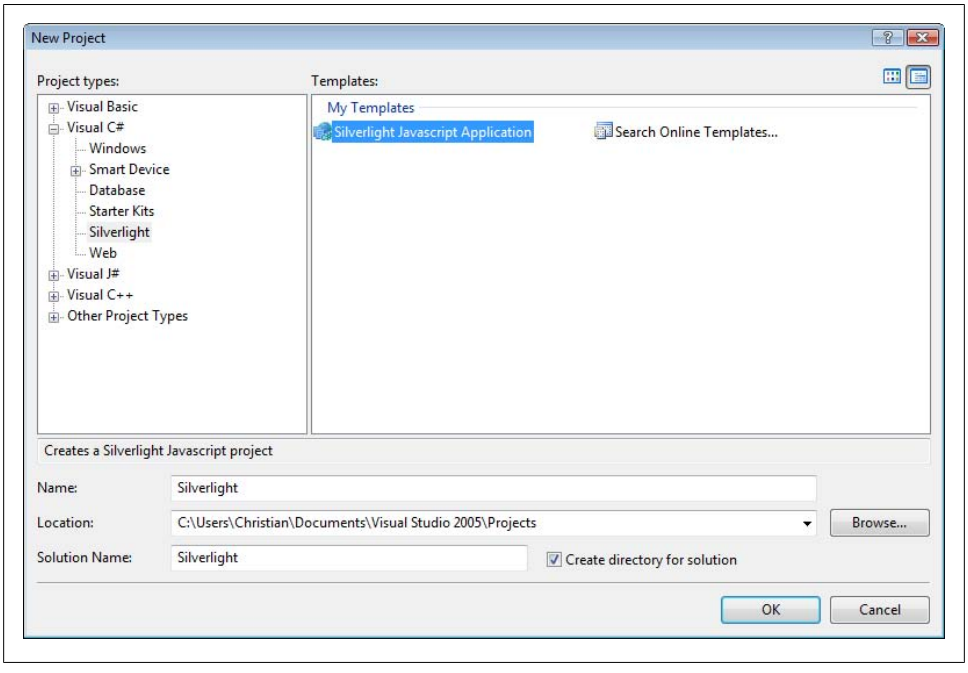


Figure 2-10. The new Silverlight templates in Visual Studio

Figure 2-11. The plugin is missing (Safari on Mac OS X)

plicitly where to put the content. This is done in the *Default.html.js* file, which is shown in Example 2-2.

Example 2-2. The JavaScript file that loads the Silverlight content (Default.html.js)

```
function createSilverlight()
{
```

Figure 2-12. The Silverlight installer for Windows

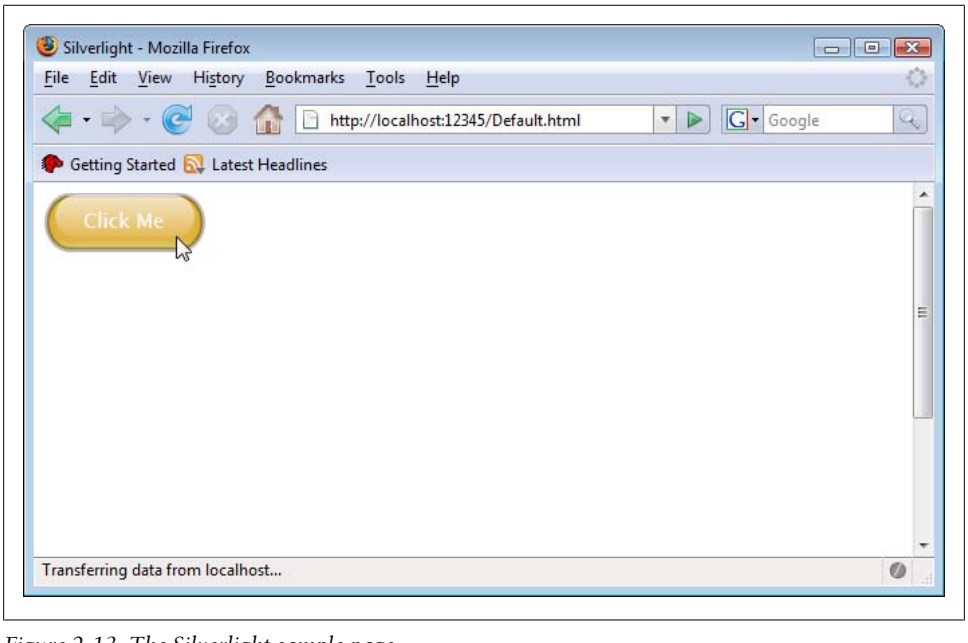


Figure 2-13. The Silverlight sample page

```
var scene = new Silverlight.Scene();
Silverlight.createObjectEx({
  source: 'Scene.xaml',
  parentElement: document.getElementById('SilverlightPlugInHost'),
  id: 'SilverlightPlugIn',
  properties: {
    width: '400',
```

```
        height: '400',
        background:'#ffffffff',
              isWindowless: 'false',
        version: '1.0'
      },
      events: {
        onError: null,
        onLoad: Silverlight.createDelegate(scene, scene.handleLoad)
      },
      context: null
    });
}

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

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

As you can see, `createSilverlight()` is defined in *Default.html.js*. It first instantiates the `Silverlight.Scene` object, which we do not need at the moment, then executes the `Silverlight.createObjectEx()` method. This method is solely responsible for initializing and loading XAML content with the help of the browser plugin. The method expects an object as an argument that holds all relevant information. The syntax of the object notation used (JSON (JavaScript Object Notation) is part of the JavaScript language syntax) is as follows:

```
{property1: "value1", property2: "value2", ...}
```

The following properties are essential:

`parentElement`
: The ID of the `<div>` container where the Silverlight content will be shown on the page

`source`
: The URL of the XAML file to be loaded

`id`
: An identification for Silverlight content that will later facilitate JavaScript access to Silverlight

`properties`
: A set of properties, including the dimensions of the content (`width`, `height`), and the background color (`background`)

`version`
: The Silverlight plugin version is at least required to run the example

The `events` property is used to wire up events, which will be covered in greater detail in Chapter 5. However, you currently need the `onError: null` setting, which will avoid

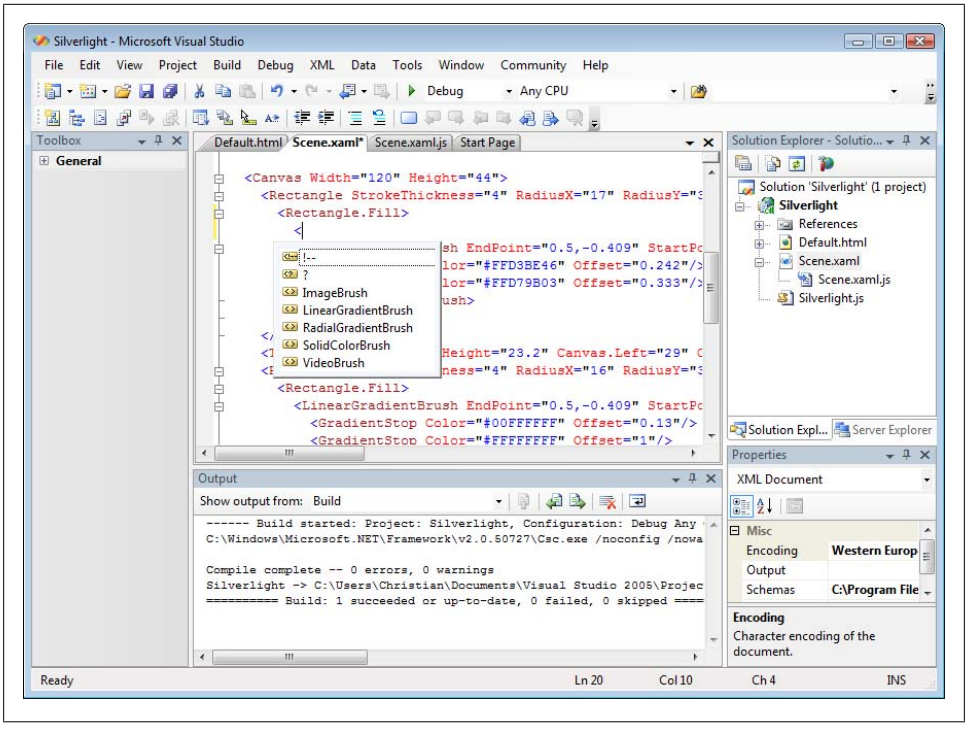


Figure 2-14. IntelliSense supports Silverlight XAML markup

a JavaScript error. Since this first example contains a bit too much complexity for a simple "Hello World," we will now create our own first Silverlight application. Create a new XML file and call it *HelloWorld.xaml*. The root element of every Silverlight XAML file is `<Canvas>` (may be compared to a `<div>` element in HTML or to ASP.NET's `<asp:Panel>`). In there, you will put an orange rectangle (`<Rectangle>`) element and a text block (`<TextBlock>` element). In the end, your markup should look like Example 2-3.

Example 2-3. A simple Hello World XAML file (HelloWorld.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15" />
  <TextBlock FontFamily="Arial" FontSize="56" Canvas.Left="25" Canvas.Top="40"
             Foreground="Black" Text="Silverlight" />
</Canvas>
```

IntelliSense will help as you type along, as Figure 2-14 shows.

The next file to be created is *Default.html.js*, which will contain the `createSilverlight()` function. You can use the *Default.html.js* file as a basis and remove all unnecessary elements (for instance the extra JavaScript code related to the scene). Don't forget to

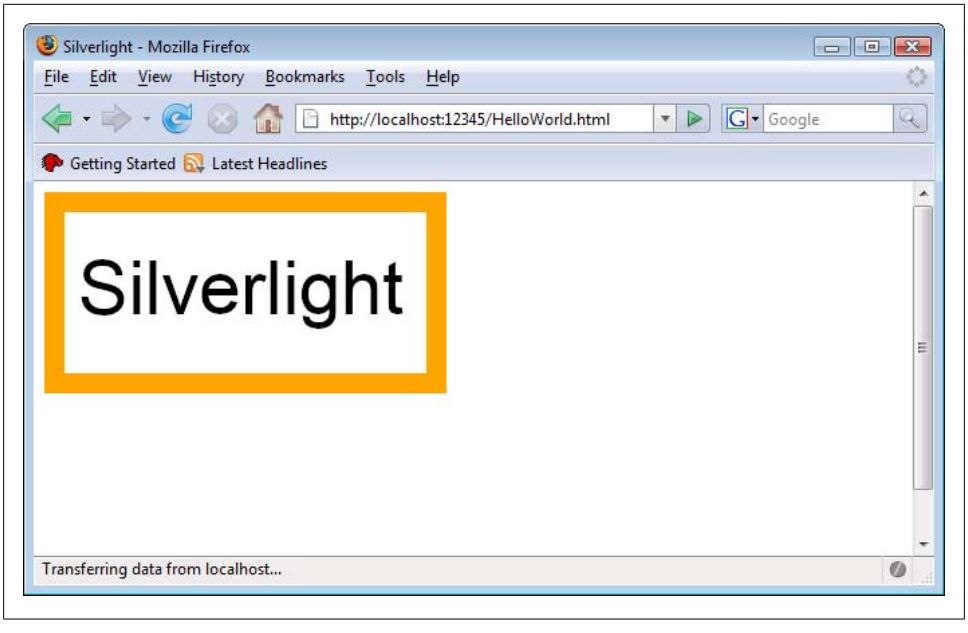


Figure 2-15. The Hello World Silverlight application in a web browser

provide the correct XAML URL in the `source` property. At the end, you should have something similar to Example 2-4.

Example 2-4. A simple Hello World XAML file (HelloWorld.xaml)

```
function createSilverlight()
{
  Silverlight.createObjectEx({
    source: 'HelloWorld.xaml',
    parentElement: document.getElementById('SilverlightPlugInHost'),
    id: 'SilverlightPlugIn',
    properties: {
      width: '400',
      height: '300',
      background:'#ffffffff',
      isWindowless: 'false',
      version: '1.0'
    },
    events: {
      onError: null,
    }
  });
}
```

Running this example in the browser will work like a charm. If you still don't have the Silverlight plugin installed, you will be prompted to do so, and then you will see the text and the orange rectangle, just as in Figure 2-15.

Figure 2-16. A new Silverlight version is available

The only file we haven't looked at yet is *Silverlight.js*. This JavaScript library takes care of several things: it tries to detect the web browser (unfortunately, it has the same habit as ASP.NET AJAX and only accepts Firefox of all the Mozilla browsers, e.g., Netscape, SeaMonkey, and others), provides the `Silverlight.createObjectEx()` method, and helps access the Silverlight content using a JavaScript API (see Chapter 8 for more information). Just copy the *Silverlight.js* file into all Silverlight applications to have this functionality.

The adoption rate of new Silverlight versions should be quite high. By default, Silverlight checks once a day to see whether there is a new version (as long as the user visits a site with Silverlight content). If there is a new version, the user is prompted to download and install the new plugin (see Figure 2-16 for the update dialog on Mac OS X); depending on the operating system and configuration, this might even happen automatically.

Further Reading

http://silverlight.net/GetStarted/
: All the resources you need to get started with Silverlight

http://silverlight.net/quickstarts/
: Silverlight quickstarts that touch upon many features

CHAPTER 3

Silverlight Tools

Why Tools?

As you have seen in the previous chapter, there is no kind of compilation or binary data with Silverlight 1.0. All you need to create are the three kinds of files:

- XAML files with the Silverlight content
- JavaScript files with both additional code for the Silverlight content and with code to control and access the Silverlight content
- HTML files to include the Silverlight content

So generally, an XML editor would be enough to create SIlverlight applications. But that's the same thing as saying "ASP.NET 2.0 applications can be created in Notepad." Of course that's possible, but who would want to do that?

Therefore, we will introduce three kinds of editors in this chapter that should be helpful to create Silverlight applications. For each kind of editor, we briefly introduce one specific editor and also have a look at the competition, if there is any.

XML Editors

For editing XAML, a mighty XML editor would be good enough theoretically. And there are many good XML editors out there, including <oXygen/> (*http://www.oxygenxml.com/*) and XMLSpy (*http://www.xmlspy.com/*). Even Microsoft has a dedicated XML editor now, XML Notepad 2007 (available at *http://www.microsoft.com/downloads/details.aspx?FamilyID=72d6aa49-787d-4118-ba5f-4f30fe913628&DisplayLang=en*; see Figure 3-1). Thanks to the availability of an XML schema for XML, code-completion is at least theoretically possible. As you have seen in the previous chapter, Visual Studio 2005 offers IntelliSense, as well, and you get even more IntelliSense (including JavaScript code) in Visual Studio 2008.

The one thing pure XML editors are always lacking is WYSIWYG functionality. Many of them try to use CSS and/or XSLT to convert the XML into something viewable

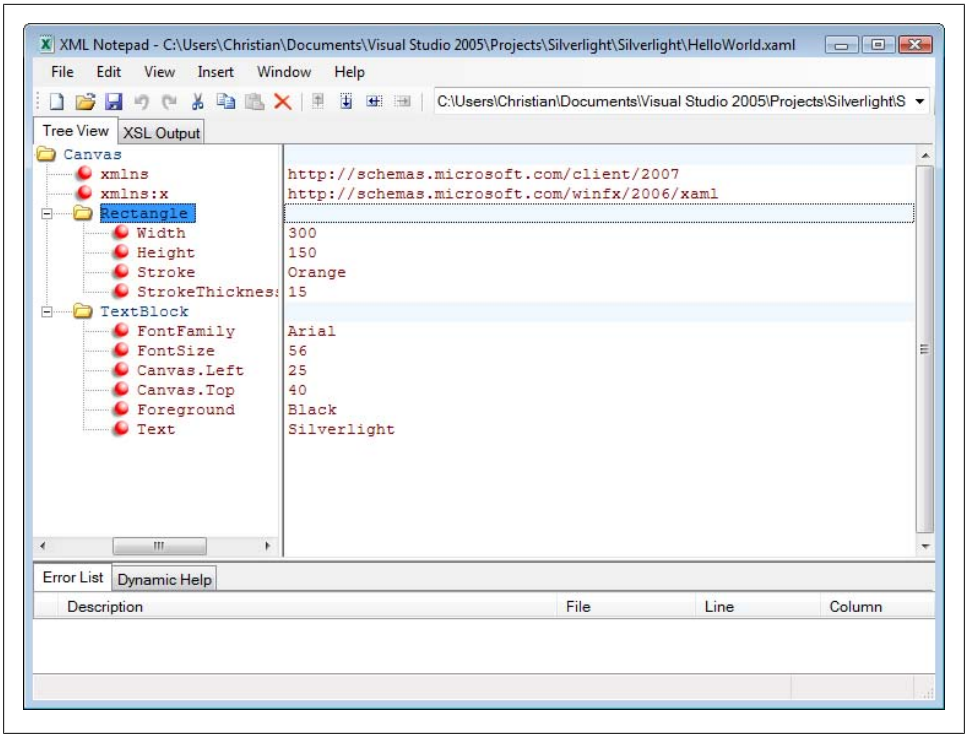


Figure 3-1. XML Notepad 2007

(including the XML editor I am writing this book in), but for Silverlight that's not a feasible option. An editor that does both is, of course, a better option. You may want to have a look at Spket IDE from *http://www.spket.com/*. It is free for non-commercial use and offers both JavaScript and XAML code completion. Figure 3-2 shows the XAML editor of Spket IDE in action.

Vector Graphics Editors

Silverlight graphics are vector-based, compared to regular web sites where images are usually pixel-based. In all fairness, Silverlight also supports pixelized graphics, but vector graphics have some advantages: no quality loss when scaling, for instance. There are several vector graphics editors, but very few of them currently support XAML. Microsoft Design was one of the first to do so and it is part of the Microsoft Expression Studio. At *http://www.microsoft.com/expression/products/download.aspx?key=design* you will find more information about it and download a timebombed trial (60 days). Microsoft Design can import a series of other formats and also export to a few, including XAML, as Figure 3-3 shows.

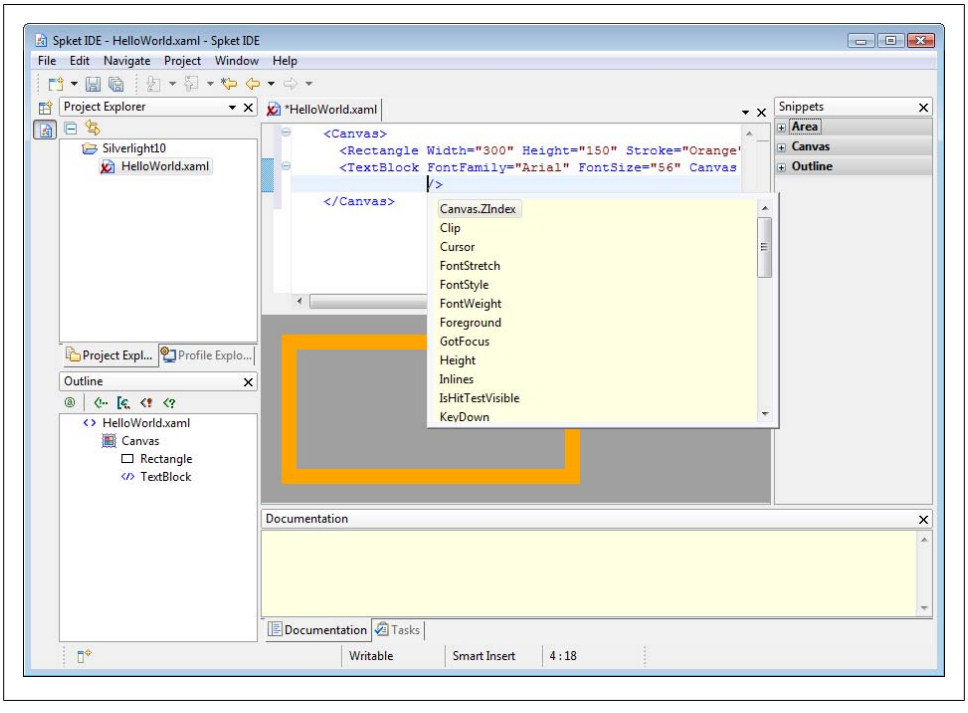


Figure 3-2. Spket IDE

Silverlight IDEs

With Flash, the name refers to the browser plugin (and the associated file format) and for the mighty editor or IDE that allows creation of these Rich Internet Applications (RIAs). So it was just a matter of time until Microsoft would release a similar tool that tries to join the bridge between designers and developers and appeals to both groups.

Microsoft Expression Blend is also part of the Microsoft Expression Studio. The first version targeted WPF developers and did not offer anything for Silverlight developers (or WPF/E developers, as they were called then). However, Expression Blend 2 changes that. (It is, at the time of this writing, available as an August Preview at *http://www.microsoft.com/expression/products/download.aspx?key=blend2preview*.)

When you set up a new project, one of the options is to create a Silverlight JavaScript application (see Figure 3-4). If you closely look at the project structure in Figure 3-5, you will see that it looks quite similar to the project based on the Silverlight Visual Studio template you saw in Chapter 2.

Expression Blend 2 is somewhat integrated with Visual Studio. Double-clicking on a JavaScript file in the project explorer opens it in Notepad (the Windows editor), but when working with event handlers (see Chapter 5 for more technical information about

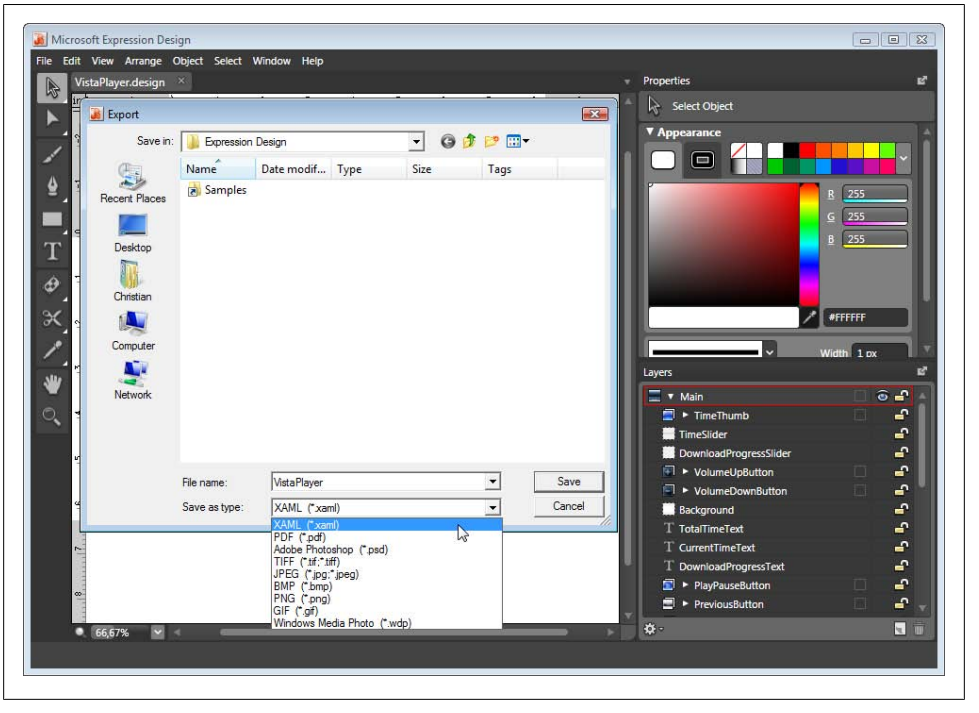


Figure 3-3. Microsoft Expression Design

that), you can set how Visual Studio 2005 handles them, as Figure 3-6 shows. Alternatively, Expression Blend 2 can also provide the skeleton code for event handlers in the clipboard, so that you can use them in any other arbitrary application. It might possible that future versions of Expression Blend 2 will facilitate integrating external applications.

Visual Web Developer 2005 Express Edition may also be used as event handler code editor.

While Expression Blend 2 is still far from being perfect (Adobe Flash had several years of time raising the bar), it is currently the best choice for Silverlight developers to get some visual help developing their applications, especially from a designer perspective.

Further Reading

http://www.microsoft.com/expression/
: An overview of Microsoft's Expression line of products

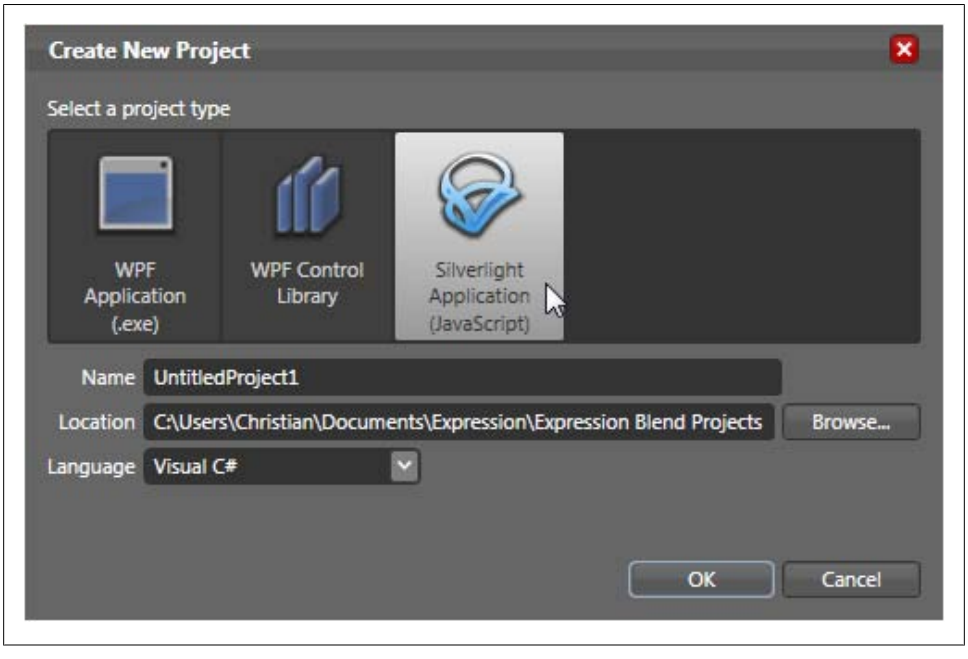


Figure 3-4. Creating a Silverlight JavaScript application within Blend 2

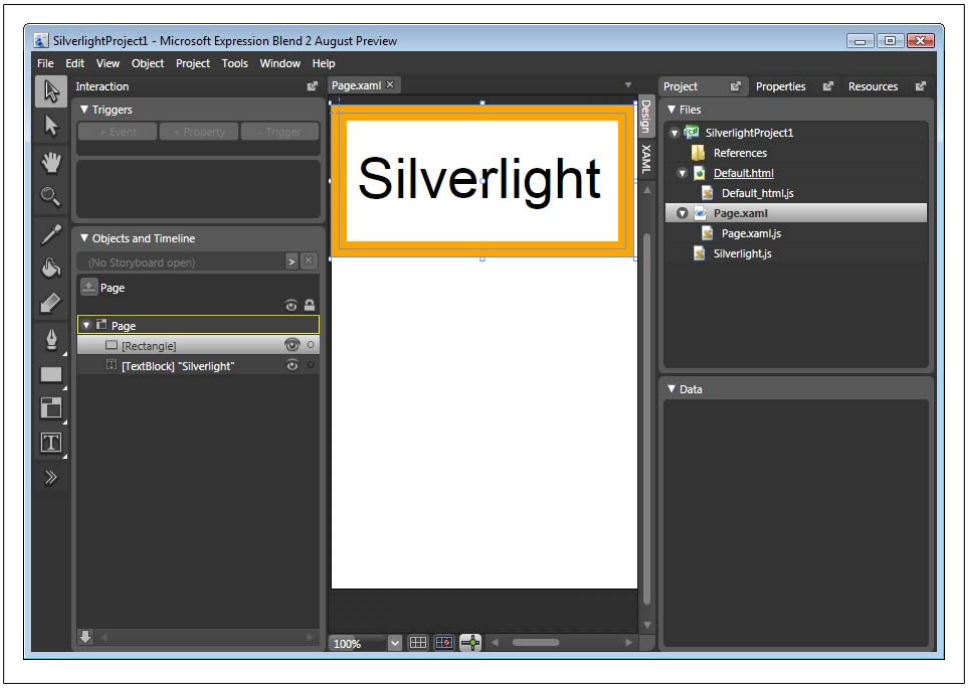


Figure 3-5. Microsoft Expression Blend 2 (August Preview, your mileage may vary)

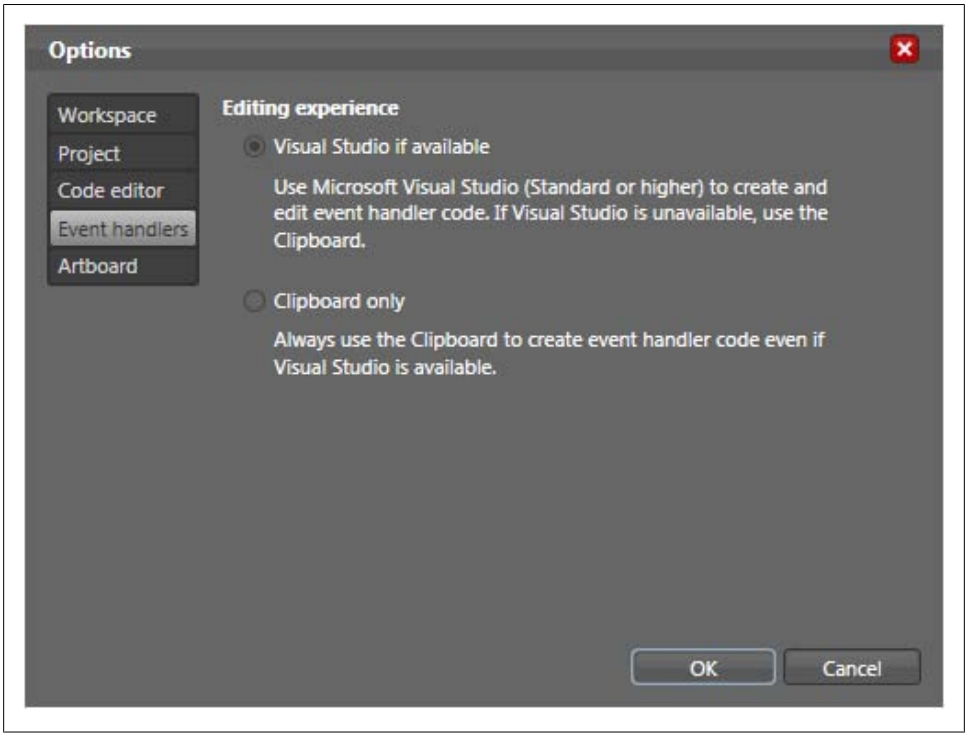


Figure 3-6. Visual Studio 2005 can take over event handling code for Silverlight applications

PART II

Declarative Silverlight

CHAPTER 4

XAML Basics

XAML

XAML is an XML dialect, so we will use a lot of angle brackets throughout this book. In this chapter, we will have a look at the most important XAML elements. It is virtually impossible to cover them all in a book of this size, but we will present as many as possible to let you dive into XAML with maximum speed.

If you have already worked with XAML for WPF applications, you already know most of what is covered in this chapter (and most of Chapter 6, as well). However, there are some subtle differences: Silverlight does not support the full XAML format like WPF, but only a subset. Future versions of Silverlight will increase the percentage of supported elements and attributes, but some things just cannot work in a web browser as they do in a desktop application.

The root element of every XAML file is `<Canvas>`, which defines the area that holds the Silverlight content. "Positioning Elements will show other uses for the `<Canvas>` element. For now, just remember to put this element at the beginning of each XAML file and supply the correct namespaces as follows:

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

Using Text

The first example used for most technology is a variation of the Hello World example (see Chapter 2 for such an example). This chapter will start with something like Hello World as well: we will add text to the Silverlight content. The element used for this is `<TextBlock>` (which you have already seen in Chapter 2), and there are two ways to provide this text:

- Within the `Text` attribute of the element

- As a text node within the element

Example 4-1 uses the latter approach to output a simple text. Note that you will get a notice in Visual Studio that using text within `<TextBlock>` is not allowed, but Figure 4-1 proves that it works.

Example 4-1. Using simple text, the XAML file (Text1.xaml)

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

To repeat the structure of a Silverlight application from Chapter 2, you need two more files to make this example work in a web browser. First, you need a helper JavaScript file that initializes the Silverlight content, such as shown in Example 4-2. Since this JavaScript file is tied to an HTML file, it is dubbed "HTML code-behind" throughout this book. In any example captions, we refer to the file as the "HTML JavaScript file" (as opposed to "XAML JavaScript files," which will be introduced in the next chapter).

Example 4-2. Using simple text, the HTML JavaScript file (Text1.html.js)

```
function createSilverlight()
{
  Silverlight.createObjectEx({
    source: 'Text1.xaml',
    parentElement: document.getElementById('SilverlightPlugInHost'),
    id: 'SilverlightPlugIn',
    properties: {
      width: '400',
      height: '300',
      background:'#ffffffff',
      isWindowless: 'false',
      version: '1.0'
    },
    events: {
      onError: null,
    }
  });
}
```

Note the highlighted code elements:

- The `source` property must be filled with the URL of the XAML file
- The `parentElement` property must be filled with a reference to the DOM element that will hold the Silverlight content
- The `id` property provides a value that JavaScript code may use to access the Silverlight content (see Chapter 8 for details)

Second, an HTML file is used as the primary page to be loaded in the browser. This file includes both the "HTML code-behind" file and the `Silverlight.js` helper file that is installed as part of the Silverlight SDK Visual Studio plugin (and is also part of the

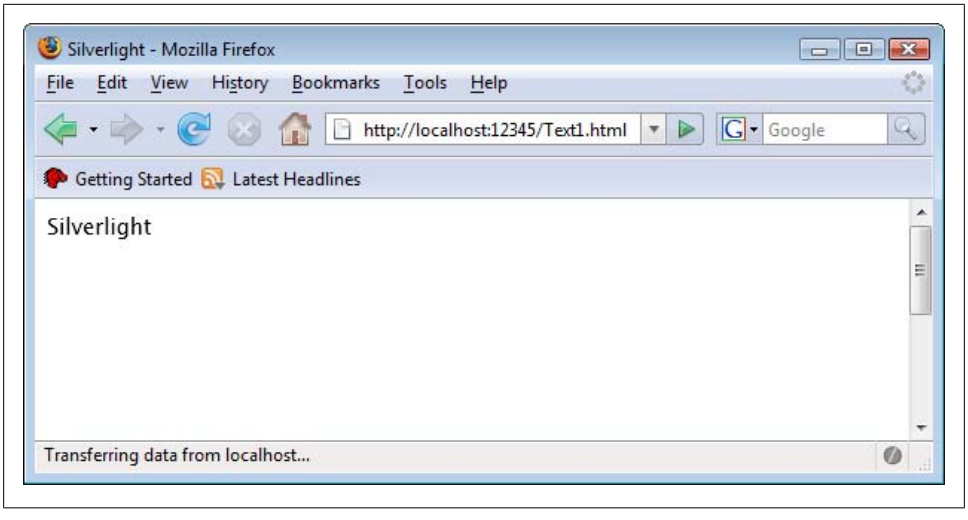


Figure 4-1. The text is displayed

downloads for this book , available at *http://www.oreilly.com/catalog/9780596516116*). The HTML page needs to contain a `<div>` container with the same ID that has been provided in the `parentElement` property. Finally, the page needs to call the previously defined `createSilverlight()` function. Example 4-3 has the full code, and Figure 4-1 shows the output—the text appears.

Example 4-3. Using simple text, the HTML file (Text1.html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Silverlight</title>

  <script type="text/javascript" src="Silverlight.js"></script>
  <script type="text/javascript" src="Text1.html.js"></script>
</head>

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

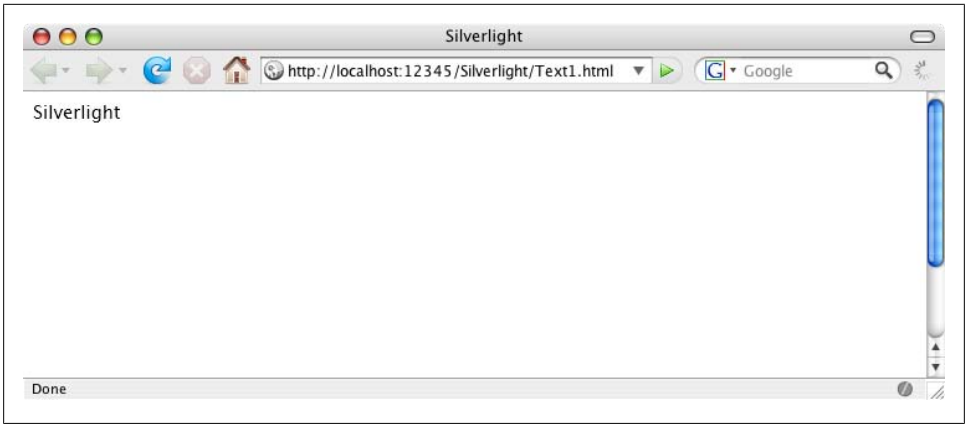


Figure 4-2. The same text on Mac OS X

So, creating new Silverlight apps starts with copying and pasting most of the time. When creating new content, you need copies of the HTML file, the HTML JavaScript file, the XAML file, and, optionally, the XAML JavaScript file. Then you just have to update all file names and you are set. Therefore, we will only print the HTML file if it is beneficial to better understand the concept of a given example. We will also avoid reprinting the HTML JavaScript file if there is no special additional information in it. The code downloads for this book always come with complete, running code.

Figure 4-1 shows the default layout for text: the text uses the Lucida font, has a size of 11 points, and is displayed in black. To make this possible, the font does not even have to be installed on the client (or on the server), it is part of the plugin. Therefore, the experience on Mac OS X is almost the same, as Figure 4-2 shows.

Apart from the Lucida font, several other fonts are also supported cross-platform:

- Arial
- Arial Black
- Comic Sans MS
- Courier New
- Georgia
- Times New Roman
- Trebuchet MS
- Verdana

Other fonts, even if they are installed on the client, are not supported; Silverlight uses Lucida if the font name is invalid.

There are several ways to apply these fonts. First of all, some of the `<TextBlock>` attributes come in handy:

`FontFamily`
: The font family name (e.g., Arial)

`FontSize`
: The font size in points (e.g., 12)

`FontWeight`
: How to display the font (e.g., `Thin`, `ExtraLight`, `Light`, `Normal`, `Medium`, `SemiBold`, `Bold`, `ExtraBold`, `Black`, and `ExtraBlack`; unfortunately, IntelliSense provides you with additional, invalid choices)

You can easily apply these attributes to a `<TextBlock>` element. However, if you would like to use different formattings in one `<TextBlock>`, you have another option. Use the `<Run>` element within `<TextBlock>` to provide inline formatting options. This concept can be compared to HTML: imagine `<TextBlock>` as a `<div>` element and `<Run>` as a `<span>` element within that `<div>` element. The styles of the `<div>` element provide the basic layout of the text within, but `<span>` styles may override `<div>` styles.

Example 4-4 shows some styling options. It also introduces one new XAML element:

The `<LineBreak>` *element*
: This element defines, well, a line break.

The `Foreground` *attribute*
: This defines the foreground (here it is text) color. You can either use a defined color name (`Red`, `Green`, `Blue`, etc.), or an RGB tripled (`#ff0000`, `#00ff00`, `#0000ff`, ...), or you use aRGB. The "a" stands for alphatransparency: Just provide a value between 0 (`00`) and 255 (`ff`) that defines the degree of the nontransparency. If you set it to `00`, the element is fully transparent (e.g., the background is seen, the element is not). If you set it to `ff`, the element is not transparent at all, so you do not see the background. If you use a value in between, the background shines through at the given degree. For instance, `#7fffff00` is a yellow (`ff0000`) that is about 50 percent transparent (`7f` is hex for 127).

You can also provide the background color for an element, using the `Background` property.

Refer to Figure 4-3 for the output in the browser.

Example 4-4. Text styling options, the XAML file (Text2.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <TextBlock Foreground="Blue" FontFamily="Arial" FontSize="24" FontWeight="Bold">
    Arial, 24pt, Bold, Blue
```

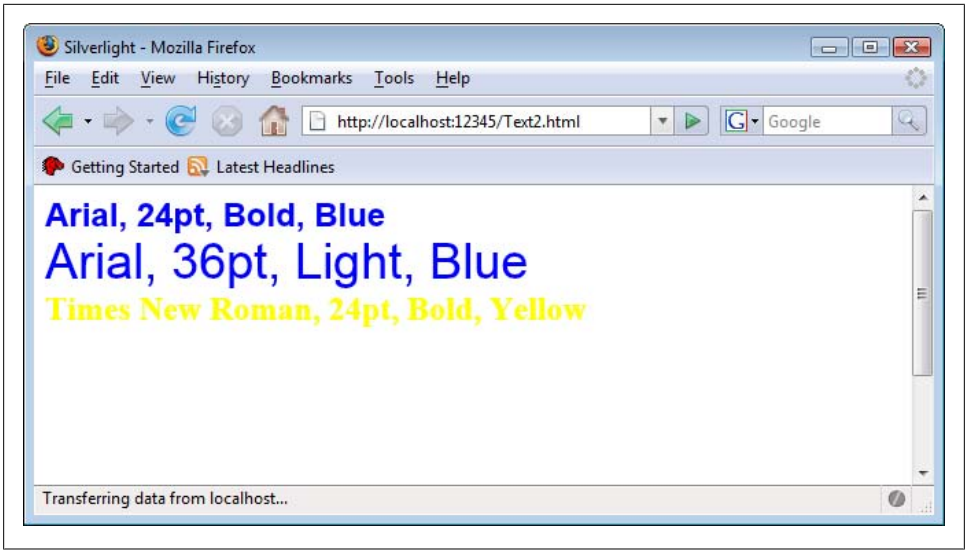


Figure 4-3. Different text styling options

```
        <LineBreak />
        <Run FontSize="36" FontWeight="Light">Arial, 36pt, Light, Blue</Run>
        <LineBreak />
        <Run FontFamily="Times New Roman" Foreground="#7fffff00">Times New Roman,
          24pt, Bold, Yellow</Run>
      </TextBlock>
    </Canvas>
```

It is possible to load external OpenType or TrueType (TTF) fonts and use them within a Silverlight application. Refer to Chapter 9 for details.

Wrapping Text

By default, the text contained in a `<TextBlock>` element does not wrap. However, by setting the `TextWrapping` property to `Wrap`, you can instruct Silverlight to automatically wrap the text for you. This of course makes most sends if you provide a fixed width for the text. For example:

```
<TextBlock Width="200" TextWrapping="Wrap"
           Text="This text will not fit in one line." />
```

Setting the `TextWrapping` property to `NoWrap` would disabled text wrapping, which is the default anyway.

Using Shapes

Most typical Silverlight visual elements are shapes: geometrical elements that make up the visual experience of the application. This section will cover many of the available options.

Before we dive into the different supported shapes, we examine formatting options. There are several of them, and many of them are specific to certain shapes, but the following three properties are shared among all shapes:

`Fill`
: How to fill the inner area of a shape, e.g., by providing a color

`Stroke`
: How to paint the outline of a shape, e.g., by providing a color

`StrokeThickness`
: The width of the outline, in pixels (must not be an integral value)

We start with probably the easiest shape: a line, represented in XAML by the `<Line>` element. You need to provide the start and end point of the line and use the Silverlight coordinate system (which is pixel-based, the origin is in the top left corner). The associated attribute names are `X1`, `Y1`, `X2`, and `Y1`. Example 4-5 paints a simple triangle, using three lines, and Figure 4-4 shows the browser output. Note that thanks to the five pixel width of the strokes, the corners of the triangle are not perfect.

Example 4-5. A triangle with three lines, the XAML file (Line.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Line Stroke="Red" StrokeThickness="5" X1="200" Y1="50" X2="350" Y2="250" />
  <Line Stroke="Green" StrokeThickness="5" X1="350" Y1="250" X2="50" Y2="250" />
  <Line Stroke="Blue" StrokeThickness="5" X1="50" Y1="250" X2="200" Y2="50" />
</Canvas>
```

If you want to create a closed shape, such as a triangle, rectangle, and so on, you would be better off using the `<Polygon>` element, which combines all points. In the `Points` property, you need to provide a list of points, using this format:

```
X1,Y1 X2,Y2 X3,Y3 ... Xn,Yn
```

The rendering algorithm is as follows: the first point is connected with the second one, the second one with the third one, and so on; at some time point number `n-1` is connected with point `n`. Finally, Silverlight connects point `n` with the very first point.

If you want to omit the final step, e.g., if you create a shape that is not closed because the last point is not connected with the first one, use `<Polyline>` instead of `<Polygon>`.

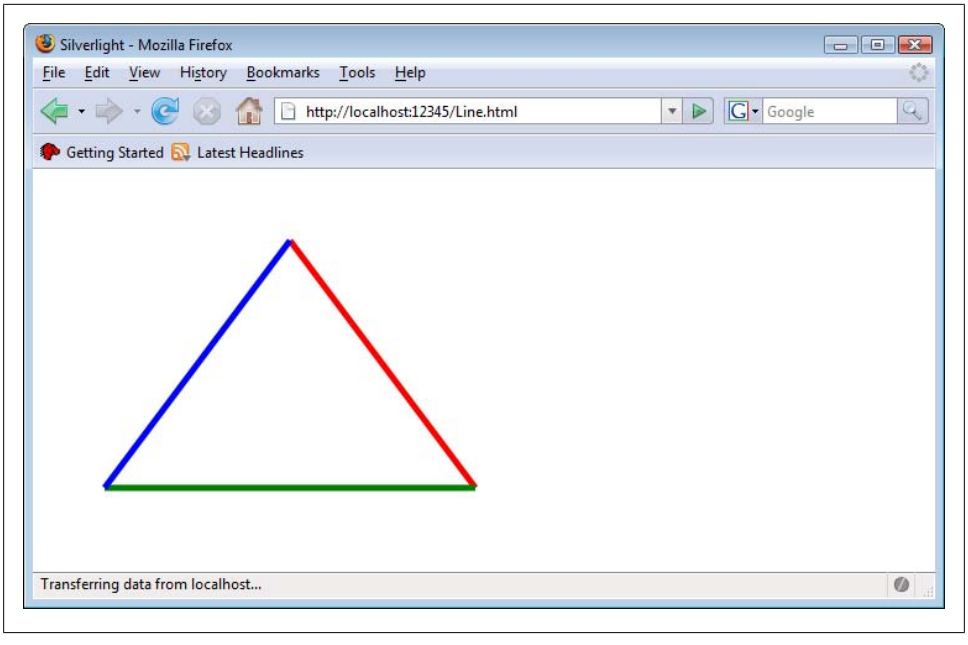


Figure 4-4. The triangle in the browser

Example 4-6 once again creates the same triangle as before, but this time the corners are much better, as Figure 4-5 shows. Since we cannot used alternating edge colors when using `<Polygon>`, we added an additional visual effect by setting the `Fill` property.

Example 4-6. A triangle as a polygon, the XAML file (Polygon.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Polygon Points="200,50 350,250 50,250"
           Stroke="Black" StrokeThickness="5" Fill="Orange" />
</Canvas>
```

A special case of a polygon is a rectangle, represented in Silverlight with the `<Rectangle>` element. Here you do not provide the coordinates of all corners (or of the top left and bottom right corner), but have a different approach: you provide the width and height of the rectangle in its `Width` and `Height` attributes. The actual position of the rectangle is provided using the technique introduced in "Positioning Elements, so we will omit this feature for now. However, we would like to showcase another feature of `<Rectangle>`: rounded corners.

A rounder corner is actually an ellipsis (which will get coverage of its own in a minute). You can now provide the radius of that ellipsis. If the horizontal and the vertical radius are the same, you get a circle, which is the most common option for a rounded corner. However, you can also provide different radius values to create a different visual effect. The attributes you need to use are `RadiusX` and `RadiusY`.

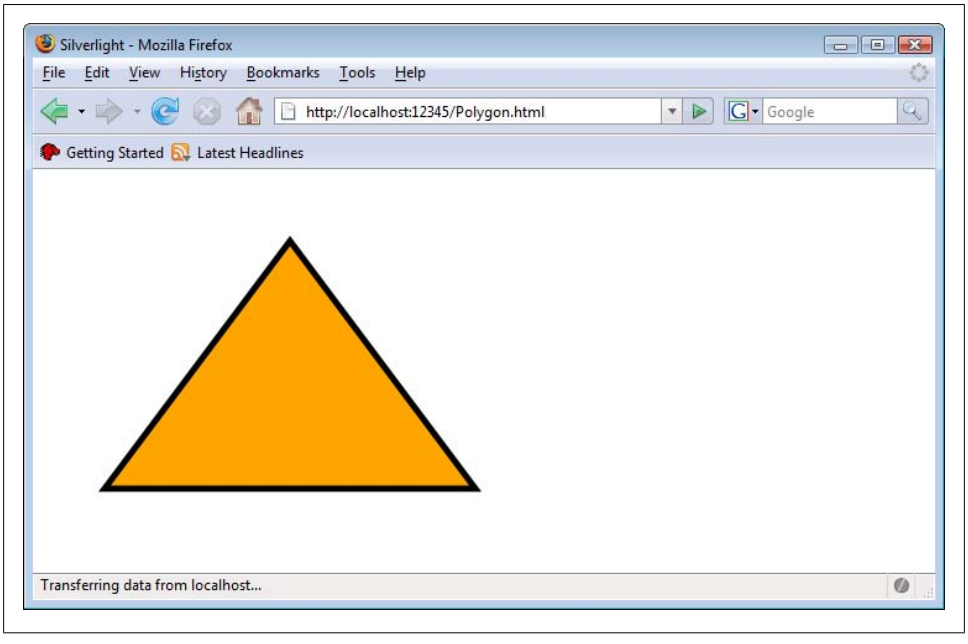


Figure 4-5. The (improved) triangle in the browser

Example 4-7 uses an ellipsis with an `RadiusX`:`RadiusY` ratio of 100:1. In Figure 4-6 you see the result: The rounded corners overlap a bit the two horizontal edges of the rectangle.

Example 4-7. A rectangle with rounded corners, the XAML file (Rectangle.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Rectangle Width="200" Height="150"
             Stroke="Black" StrokeThickness="5" Fill="Orange"
             RadiusX="100" RadiusY="1"/>
</Canvas>
```

The final shape is also the most important one, especially if you have complex designs, such as a path, or the `<Path>` element. Its most important property is `Data`, which contains information defining the path. To tell the truth, it's usually design tools that create paths, since any shape, regardless of how complex it is, can be transformed into a path. This section will provide you with a crash course of the path syntax.

A path consists of several painting instructions: moving the virtual "pen" to a certain position, drawing certain shapes, and ending the drawing. Every instruction starts with a (case-insensitive) letter that identifies the instruction and several parameters may follow.

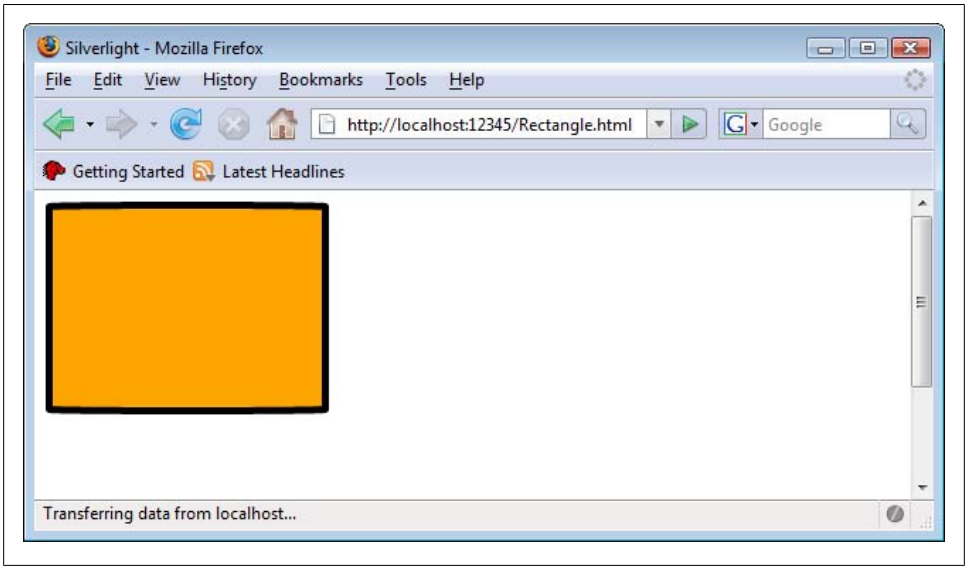


Figure 4-6. The rectangle with the rounded corners

The first part of a path is the so-called fill rule. This takes care of a special case: what happens if elements in the path overlap. You may choose between the default value `F0` and `F1`:

`F0`
: Stands for `EvenOdd`, meaning that points that have an even number of path segments between them and the end of the canvas are considered outside the path; points with an odd number are considered inside and would be filled.

`F1`
: Stands for `NonZero`, meaning that all points where a line between the point and the end of the canvas crosses the path from the left side as often as from the right side are considered inside the path.

Generally, `EvenOdd` is what you will want, and since it is the default value, you do not have to provide it at all.

Next up are the instructions. The first one is usually `M`, which stands for "move." This moves the virtual pen to a certain position but does not start drawing. The following path would put the pen at the x coordinate 40 and the y coordinate 30:

```
M 40,30
```

Starting from that point, several shapes are possible. We will start once again with a line, denoted by the `L` command. You only have to provide the end point of the line—the starting point is defined by the current pen position! The following path would therefore draw a line from (40,30) to (70,80):

```
M 40,30 L 70,80
```

Special cases of lines are horizontal lines (`H` command) and vertical lines (`V` command). For horizontal lines you only need to provide the x coordinate of the end point; for vertical lines you only need to provide the y coordinate of the end point.

By using lines, you can create any geometric shape that does not have curves. For curves, however, several options exits. The `A` command draws an elliptical arc. You need to provide a set of parameters:

- The x and y radius of the ellipsis
- The rotation angle of the ellipsis (use degrees)
- Whether the angle is larger than 180 degrees (`1`) or not (`0`)
- Whether the arc is drawn in positive direction (`1`) or not (`0`)
- The end point of the arc

The following markup would create an arc from (50,50) to (100,50), using an x and y radius of 75 each, with a 90 degrees rotation angle in positive direction:

```
M 50,50 A 50,50 90 0 1 100,50
```

A type of curve that is very common in the vector graphics field are Bézier curves, named after French automobile designer Pierre Bézier. Assume that you have two points, A and B. Bézier defined a couple of mathematical equations that define curves between those points. The easiest one is a linear curve, but they are easy to draw without any extra help from Silverlight. However, there are more complex variants. A quadratic Bézier curve (called that because in the defining formula values are squared) uses a so-called control point to shape the exact look of the curve. The associated Silverlight path command, `Q`, provides the coordinates of this control point and also of the end point; remember that the start point is again defined by the current position of the pen.

The following markup moves the pen to (125,125) and creates a Bézier curve to (175, 75), using (110, 60) as a control point:

```
M 125,125 Q 110,60 175,75
```

A cubic Bézier curve goes one step further and uses two control points. The associated SIlverlight path command is `C`. Here is an example: the curve goes from (150,125) to (50,100), using the two control points (125,175) and (20,125).

```
M 150,125 C 125,175 20,125 50,100
```

There are more advanced Bézier curves available, as well: They take the previous point of the curve into account, making the curve look more smooth. For "smooth," sister of the quadratic Bézier curve uses the `S` command, and the "smooth" cubic Bézier curve uses `T`. The syntax is the same as with the `Q` and `C` commands.

One final command is missing, it is called `Z` and closes a path, meaning that the pen draws a straight line to the beginning of the path.

Example 4-8 shows several of the previous path commands in action. In Figure 4-7, working clockwise, you can see a straight line (red), an elliptic arc (yellow) to the right of the straight line, a quadratic Bézier curve (green) to the right of the arc, and a cubic Bézier (blue) curve. The control points for the Bézier curves have been marked with an X so you can see which points the curves are approaching. These markers have been also created using a path (drawing two crossing lines).

Example 4-8. Using Paths, the XAML file (Path.xaml)

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

  <Path Data="M 20,10 L 40,70"
        Stroke="Red" StrokeThickness="5" />

  <Path Data="M 50,50 A 50,50 90 0 1 100,50"
        Stroke="Yellow" StrokeThickness="5" />

  <Path Data="M 125,75 Q 200,100 175,125"
        Stroke="Green" StrokeThickness="5" />
  <Path Data="M 195,95 L 205,105 M 205,95 L 195,105"
        Stroke="Black" StrokeThickness="2" />

  <Path Data="M 150,125 C 125,175 20,125 50,100"
        Stroke="Blue" StrokeThickness="5" />
  <Path Data="M 120,170 L 130,180 M 130,170 L 120,180"
        Stroke="Black" StrokeThickness="2" />
  <Path Data="M 15,120 L 25,130 M 25,120 L 15,130"
        Stroke="Black" StrokeThickness="2" />

</Canvas>
```

As mentioned at the beginning of this section, creating a path manually can be painful, so you should use a graphics software for that. However, you can now analyze and understand paths that are created by vector graphic programs.

If you have used SVG before, this path syntax will be very similar to what you are used to. Most vector formats use the same features for their paths, so the syntaxes are very alike.

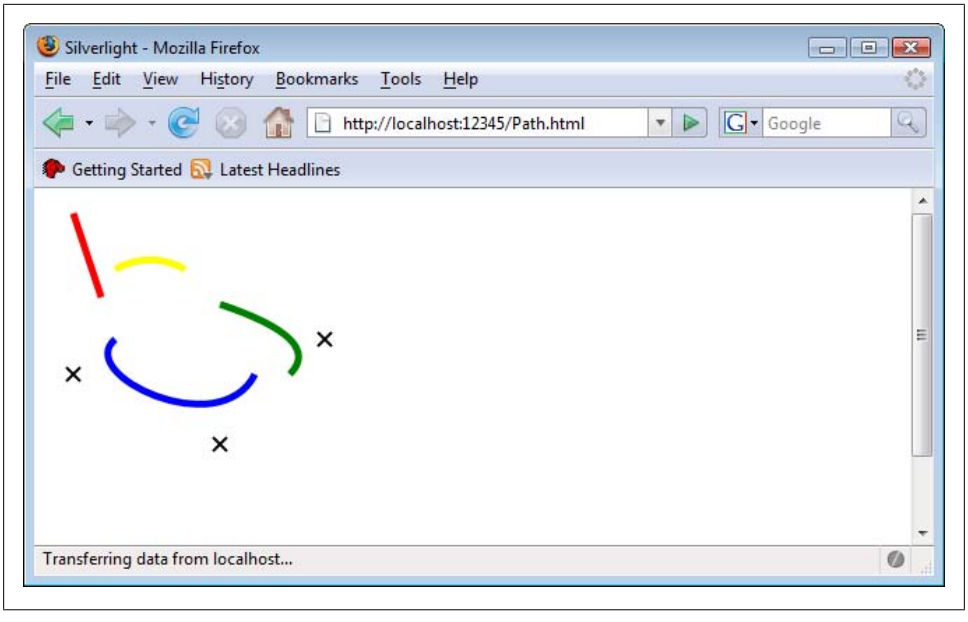


Figure 4-7. Various paths

One more shape should not be forgotten: An ellipse, represented by the `<Ellipse>` element. The most important attributes are `Width` and `Height`, defining the size of the ellipse. Example 4-9 shows such an ellipse, and Figure 4-8 has the browser output.

Example 4-9. Using an Ellipse, the XAML file (Ellipse.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Ellipse Width="400" Height="300"
           Stroke="Black" StrokeThickness="5" Fill="Orange" />
</Canvas>
```

When the ellipse has the same width and height, you get—a circle.

Geometry Elements

An alternative approach to drawing shapes is to use a so-called geometry element. It can be compared to shapes (there are lines, rectangles, paths, etc.), but this element doesn't draw itself. Instead, it can be used within other elements defining how they look. For instance, the `Clip` property of an UI object can be set to a geometry element defining a path. This path then defines the outer border of the UI object. Or, you could use a geometry element as the `Data` property of a `<Path>` element, and, therefore, provide the layout of the path.

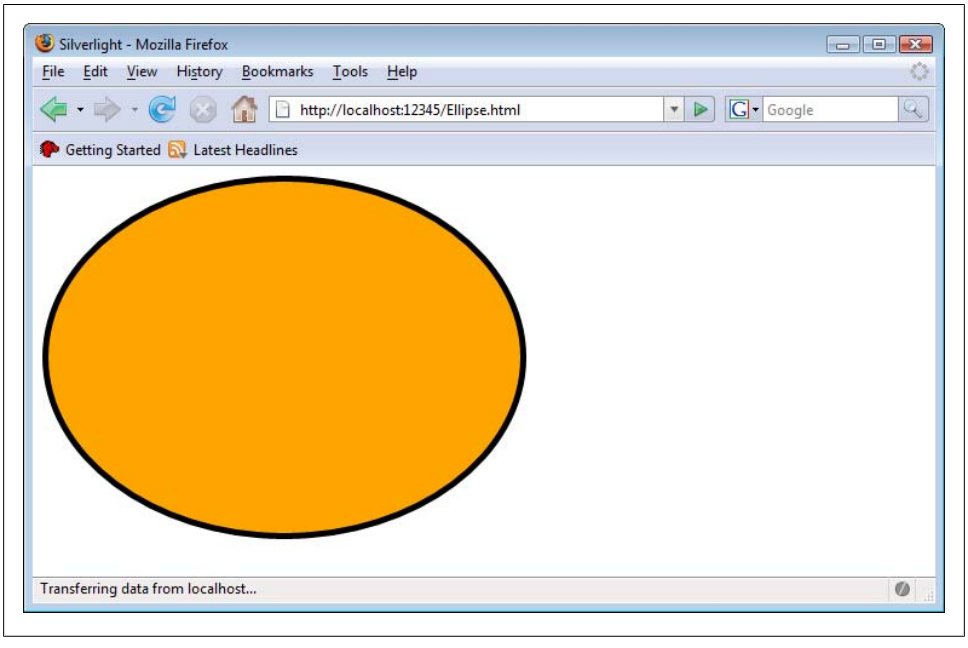


Figure 4-8. The ellipse

There are several geometry elements, including `EllipseGeometry`, `LineGeometry`, `PathGeometry`, and `RectangleGeometry`. The following is an example that clips an image by using a path (later, Example 4-16 will show a different approach to reach the same effect):

```
<Image Source="image.png">
  <Image.Clip>
    <EllipseGeometry Center="150,75" RadiusX="300" RadiusY="150" />
  </ImageClip>
</Image>
```

Geometry elements may also be combined (grouped), by nesting them under the `<GeometryGroup>` element.

Positioning Elements

If you don't specify the position of an element, it is positioned at the origin (0, 0) of the display area. You can try this out yourself: Create a Silverlight XAML file and put some `<TextBlock>` elements on it. The text contents within those elements will overlap, since all text is displayed with the top left corner at (0, 0).

This can be changed for most elements by setting their `Canvas.Top` and `Canvas.Left` properties. These properties denote the `x` and `y` coordinate of the element, respectively.

The following text block would be shown 50 pixels to the right, 100 pixels to the bottom:

```
<TextBlock Canvas.Left="50" Canvas.Top="100" Text="Silverlight" />
```

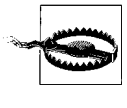
However, there is more to positioning and here the `<Canvas>` element comes into play again. A canvas can also have a position:

```
<Canvas Canvas.Left="50" Canvas.Top="100">
...
</Canvas>
```

The clue is that all elements *within* the canvas are positioned relative to the surrounding canvas. Have a look at Example 4-10, for instance. It contains several canvases, each have (except for the outer one) `Canvas.Top="50"` and `Canvas.Top="50"`. Inside the innermost `<Canvas>` element resides a `<TextBlock>` element with `Canvas.Left="50"` and `Canvas.Top="50"` as well. Each of those indentations always refer to the parent canvas and are no absolute coordinates. Therefore, each canvas starts 50 pixels to the right and 50 pixels to the bottom from where its parent canvas starts. The `Canvas.Left` and `Canvas.Top` properties are also called dependency properties: they depend on their parent `<Canvas>` element. Likewise, `<Canvas>` elements may be called dependency objects. Figure 4-9 shows the browser output.

Example 4-10. Nested, positioned canvases in the XAML file (Canvas.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Width="500" Height="500" Background="Red">
  <Canvas Canvas.Left="50" Canvas.Top="50" Background="Green"
          Width="400" Height="400">
    <Canvas Canvas.Left="50" Canvas.Top="50" Background="Yellow"
            Width="300" Height="300">
      <Canvas Canvas.Left="50" Canvas.Top="50" Background="Blue"
              Width="200" Height="200">
        <TextBlock Canvas.Left="50" Canvas.Top="50" FontSize="20"
                   Text="Silverlight"/>
      </Canvas>
    </Canvas>
  </Canvas>
</Canvas>
```



Only `<Canvas>` elements that have a fixed width and height show their background color. If you omit this information, the background remains the default, which in our example is white.

Of course these canvases overlap each other. Silverlight uses the following approach: All elements are stacked onto each other, so there is a (virtual) third dimension. Therefore, the text from Example 4-10 resides on top of all canvases, since this element comes last in the document. This is the reason why the text can be seen at all. In CSS, there

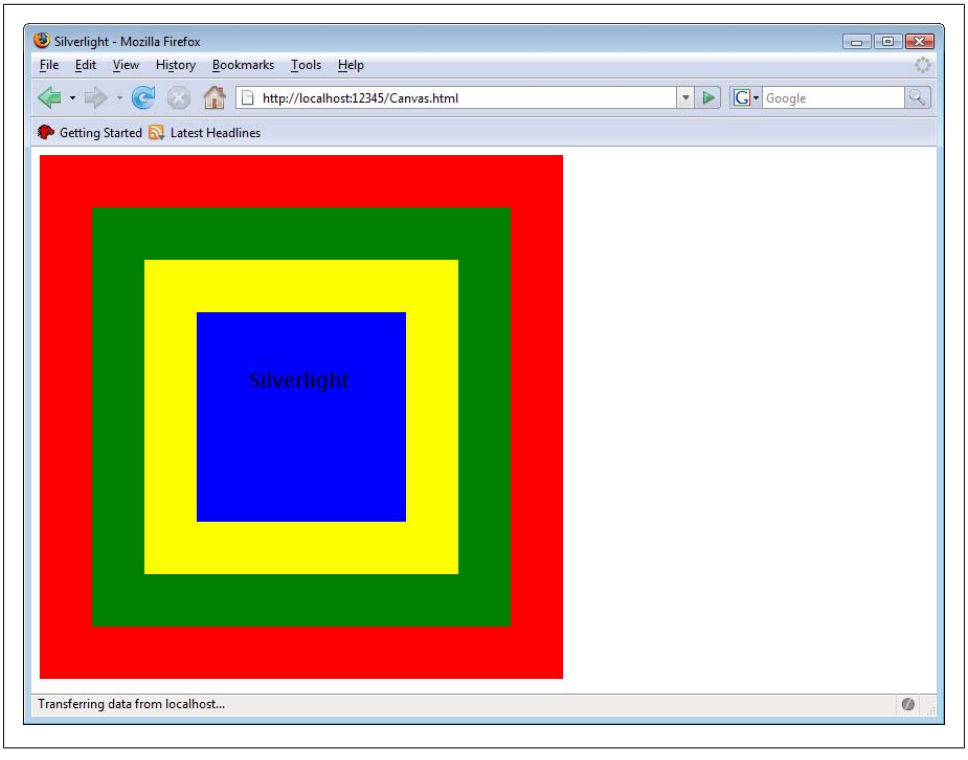


Figure 4-9. The nested canvases

is a property called `z-index` that assigns the "z coordinate" of an element: the higher the value, the further up on the stack it is.

Silverlight uses the same principle. You may assign a z-index by setting the `Canvas.ZIndex` property. Note that you can also nest z-index values; however, these values are only compared on the same element level. Assume that you have a canvas with `z-index 3` that contains two rectangles with `z-index 2` and `z-index 1`. The rectangle with the higher z-index is placed above the one with the lower z-index. However, the outer canvas will not overlap the rectangles, although its z-index is higher.

Example 4-11 is a variation of Example 4-10: everything except the outer canvas is gone, but we added rectangles. Usually they would overlap similar to Figure 4-9, but this time we set `Canvas.ZIndex` so that the "inner" elements have a lower z-index. Therefore, the first rectangle is drawn over the second one, the second one is drawn over the third one, and so on. The lowest z-index is assigned to the text block. This text block is now overlapped by the blue rectangle. Therefore, the text itself is not visible, as Figure 4-10 shows.

Example 4-11. Setting the z-index, the XAML file (ZIndex.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

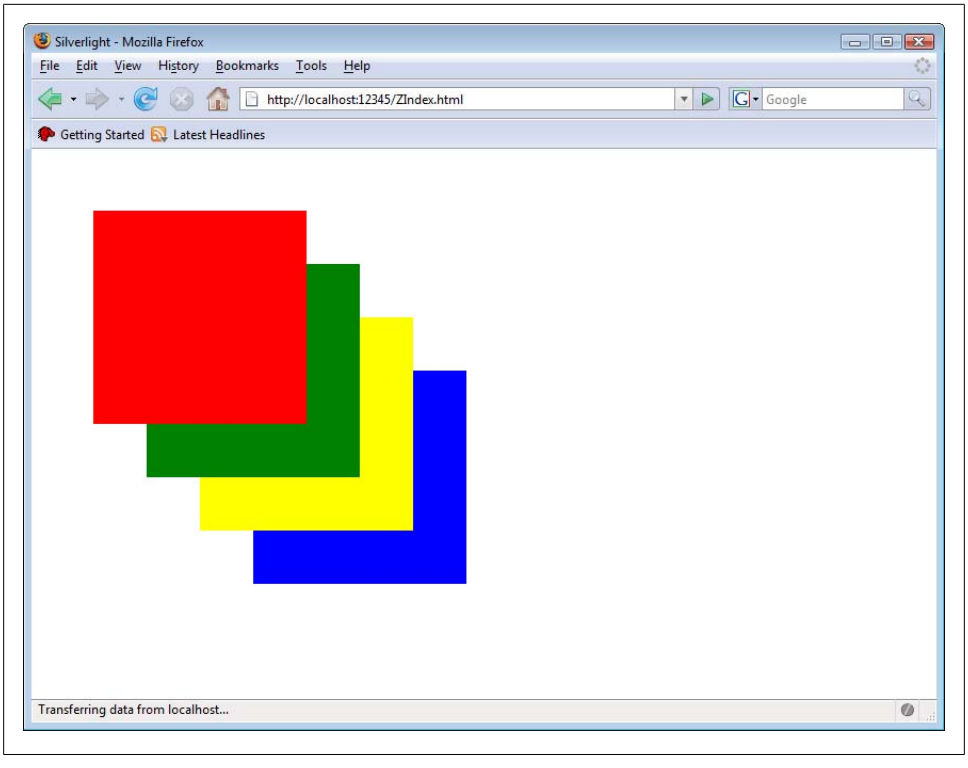


Figure 4-10. Overlapping rectangles with z-index

```
        Width="500" Height="500" Background="White">
    <Rectangle Canvas.Left="50" Canvas.Top="50" Fill="Red"
               Width="200" Height="200" Canvas.ZIndex="5"/>
    <Rectangle Canvas.Left="100" Canvas.Top="100" Fill="Green"
               Width="200" Height="200" Canvas.ZIndex="4"/>
    <Rectangle Canvas.Left="150" Canvas.Top="150" Fill="Yellow"
               Width="200" Height="200" Canvas.ZIndex="3"/>
    <Rectangle Canvas.Left="200" Canvas.Top="200" Fill="Blue"
               Width="200" Height="200" Canvas.ZIndex="2"/>
    <TextBlock Canvas.Left="250" Canvas.Top="250" FontSize="20"
               Text="Silverlight" Canvas.ZIndex="1"/>
</Canvas>
```

Using Images

Although Silverlight is a vector-based technology, pixel images are supported too. The XAML element is (conveniently) named `<Image>`. Apart from the default properties, such as `Canvas.Left`, `Canvas.Top`, `Height`, and `Width`, `<Image>` needs to know which graphics to show. This information is provided in the `Source` property. You can use

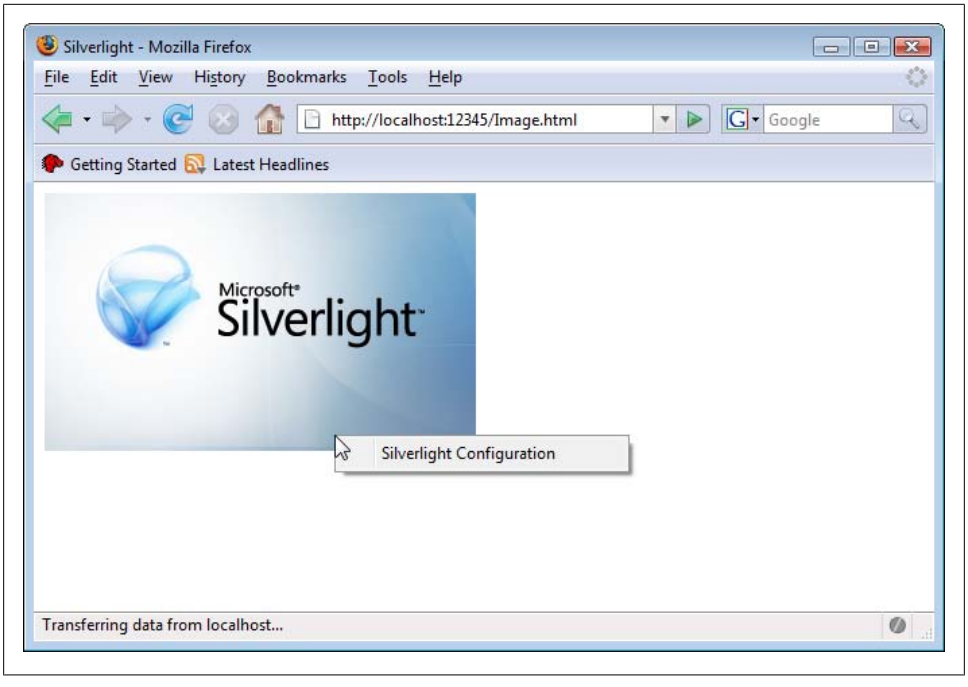


Figure 4-11. The pixel image within the Silverlight content

both local and remote URLs, and you can use two supported graphics formats: JPEG and PNG. Example 4-12 has the code, and Figure 4-11 shows the associated output.

Example 4-12. Using an image, the XAML file (Image.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Image Source="silverlight.png" />
</Canvas>
```

When using images, you can also track the data transfer using JavaScript, as Chapter 9 shows.

Using Brushes

The final basic XAML elements used to design a static (i.e., non-moving) UI are brushes. A brush is used just like a "real" brush–you can paint. However, Silverlight brushes offer more: you can paint with color, you can paint gradients, you can paint images, you can even paint videos.

Brushes can be used as alternatives to attributes such as `Background`, `Fill`, or `Stroke`. However, you need to alter your syntax a bit. Instead of using the attribute, you use a sub element, `<ElementYouWantToBrush.OldAttribute>`. For example, filling a rectangle would look as follows:

```
<Rectangle>
  <Rectangle.Fill>
    <!-- brushes go here -->
  </Rectangle.Fill>
</Rectangle>
```

The "easiest" brush is called `SolidColorBrush` because it only uses one solid color, there are no changes within the color or gradients. Actually, when using attributes like `Background` or `Fill` or `Stroke` as we have done so far in this book, we were implicitly using a `SolidColorBrush`. However, the alternative syntax works as well. Example 4-13 has the same output as Example 4-11 (Figure 4-10), but is using the `<SolidColorBrush>` element. Note how the color used by the brush is defined by its `Color` attribute.

Example 4-13. Using a solid color brush, the XAML file (SolidColorBrush.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Width="500" Height="500" Background="White">
  <Rectangle Canvas.Left="50" Canvas.Top="50" Width="200" Height="200"
             Canvas.ZIndex="5">
    <Rectangle.Fill>
      <SolidColorBrush Color="Red" />
    </Rectangle.Fill>
  </Rectangle>
  <Rectangle Canvas.Left="100" Canvas.Top="100" Width="200" Height="200"
             Canvas.ZIndex="4">
    <Rectangle.Fill>
      <SolidColorBrush Color="Green" />
    </Rectangle.Fill>
  </Rectangle>
  <Rectangle Canvas.Left="150" Canvas.Top="150" Width="200" Height="200"
             Canvas.ZIndex="3">
    <Rectangle.Fill>
      <SolidColorBrush Color="Yellow" />
    </Rectangle.Fill>
  </Rectangle>
  <Rectangle Canvas.Left="200" Canvas.Top="200" Width="200" Height="200"
             Canvas.ZIndex="2">
    <Rectangle.Fill>
      <SolidColorBrush Color="Blue" />
    </Rectangle.Fill>
  </Rectangle>
  <TextBlock Canvas.Left="250" Canvas.Top="250" FontSize="20"
             Text="Silverlight" Canvas.ZIndex="1"/>
</Canvas>
```

Brushes can do unique things. The most typical example is gradients. A common form of a gradient is a radial gradient: The gradient starts at a given origin (quite often the center of an object) and then goes radially to the borders of the object. You can define

an arbitrary number of stop points: these are points where a certain color must be matched. So, all you need to do is to define the stop points and associated colors; Silverlight automatically calculates and draws all colors in between. The XAML element for the brush is `<RadialGradientBrush>`.

There are a few parameters that must be defined for this gradient:

`Center`
: The center of the object. You need to provide values between 0 and 1 for both the x and the y coordinate. Silverlight then calculates the actual coordinates based on the dimension of the target object.

`GradientOrigin`
: The center of the gradient. Again, provide values between 0 and 1 for both coordinates.

`RadiusX`, `RadiusY`
: The x and y radius of the gradient, again as values between 0 and 1

Stop colors are defined using the `<GradientStop>` element. You need to provide the color (`Color` attribute), and the offset (`Offset` attribute, value between 0 and 1). Example 4-14 shows a radial gradient with thee stop colors, and Figure 4-12 has the output.

Example 4-14. Using a radial gradient, the XAML file (RadialGradientBrush.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="600" Height="600">
  <Ellipse Width="600" Height="600" Stroke="Black">
    <Ellipse.Fill>
      <RadialGradientBrush Center="0.5 0.5" GradientOrigin="0.33 0.67"
                           RadiusX="0.5" RadiusY="0.5">
        <GradientStop Color="Red" Offset="0"/>
        <GradientStop Color="Green" Offset="0.33"/>
        <GradientStop Color="Blue" Offset="0.67"/>
      </RadialGradientBrush>
    </Ellipse.Fill>
  </Ellipse>
</Canvas>
```

The other form of gradient is a linear gradient: The color change does not happen radially, but instead along a gradient axis. In the associated XAML element, `<LinearGradialBrush>`, you need to assign a start point and an end point, once again using values between 0 and 1, which are then mapped to the actual coordinates. Example 4-15 shows `<LinearGradialBrush>` in action, and also includes a line and markers that represent the radial axis and the stop points (trust me with the values), as you can see in Figure 4-13.

Example 4-15. Using a linear gradient, the XAML file (LinearGradientBrush.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="600" Height="600">
```

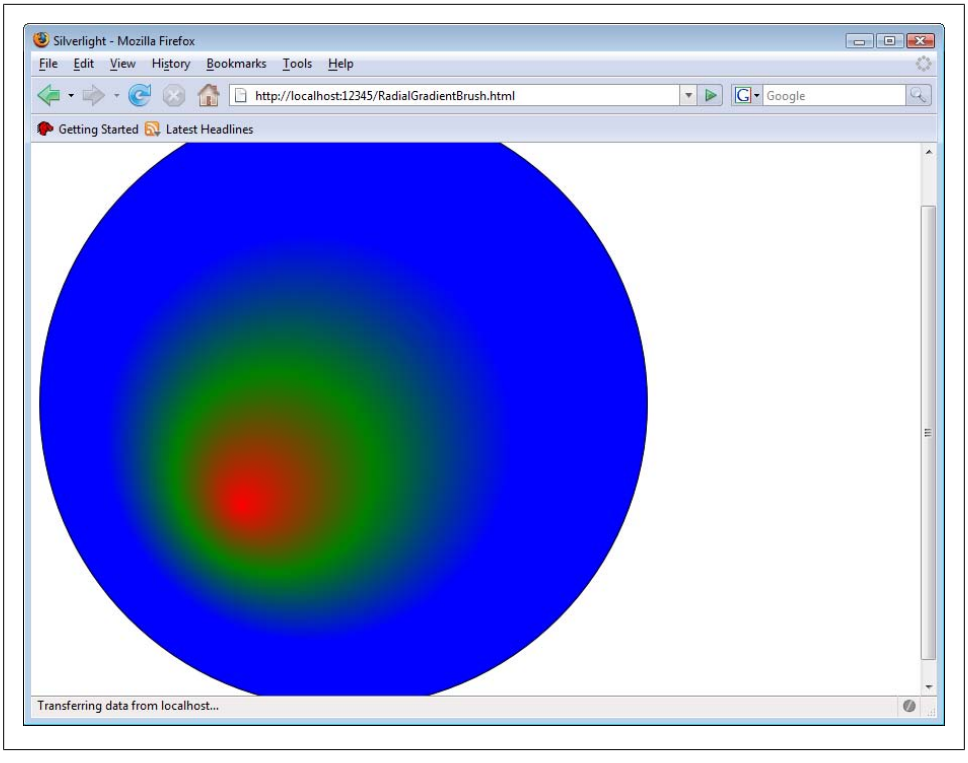


Figure 4-12. The radial gradient. Do you see the center and the stop colors?

```
    <Rectangle Width="600" Height="600" Stroke="Black">
      <Rectangle.Fill>
        <LinearGradientBrush StartPoint="0.1 0.9" EndPoint="0.9 0.1">
          <GradientStop Color="Red" Offset="0"/>
          <GradientStop Color="Green" Offset="0.33"/>
          <GradientStop Color="Blue" Offset="0.67"/>
        </LinearGradientBrush>
      </Rectangle.Fill>
    </Rectangle>
    <Path Stroke="Black" Data="M 55 535 L 65 545 M 65 535 L 55 545" />
    <Path Stroke="Black" Data="M 215 375 L 225 385 M 225 375 L 215 385" />
    <Path Stroke="Black" Data="M 375 215 L 385 225 M 385 215 L 375 225" />
    <Line X1="60" Y1="540" X2="540" Y2="60" Stroke="#7f000000" />
  </Canvas>
```

A final brush option is to use a special "filling" for a brush: an image or a video file. So when you have an image to fill an object, Silverlight can automatically stretch the content so that it fits. You could also play a video as a background for a rectangle or within an ellipsis, just to give you a few ideas.

Using both brushes, `ImageBrush` and `VideoBrush`, is quite similar. You have to provide the name of the source file in an associated attribute, which is called `ImageSource` for `<ImageBrush>` and `SourceName` for `<VideoBrush>`. You can instruct Silverlight on how to

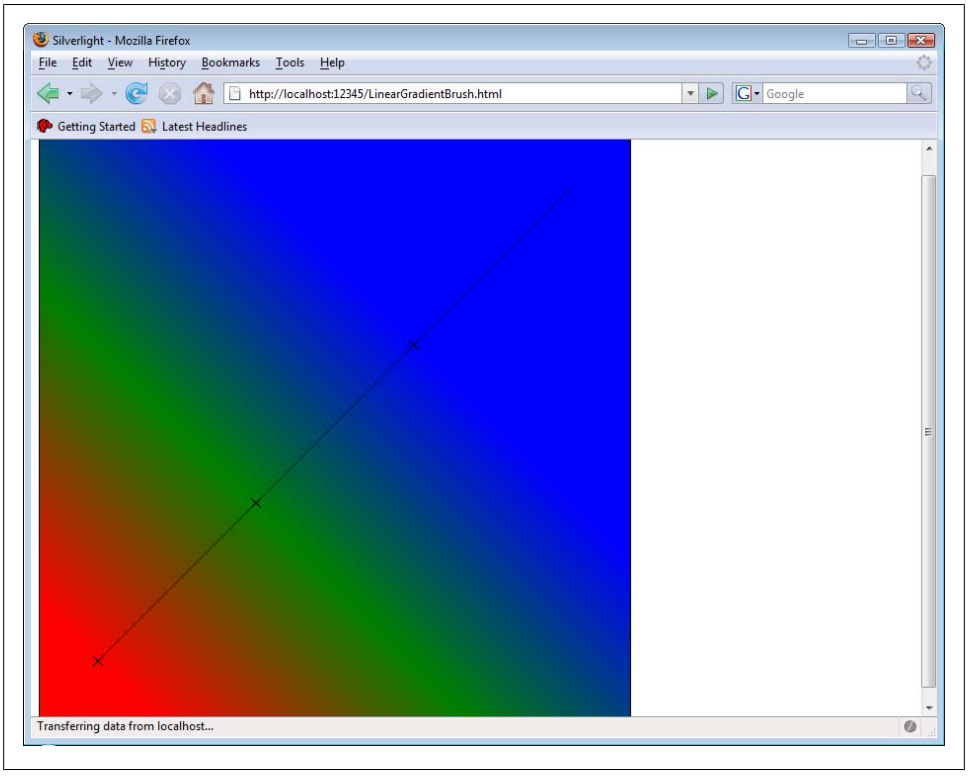


Figure 4-13. The linear gradient, with highlighted gradient axis and stop points

stretch the content so that it fits using the `Stretch` attribute, assigning one of these values:

`None`
: Content size remains the original one

`Fill`
: The content fills up the whole available area, loosing its aspect ratio

`Uniform`
: The content size is increased, maintaining the aspect ratio, until either the content has the width or the height of the display area

`UniformToFill`
: The content size is increased, maintaining the aspect ratio, until the content width and height are both greater or equal than the width and height of the display area. If necessary, parts of the content are cropped.

Example 4-16 shows an image that is used to fill an ellipsis. You can see in Figure 4-14 that this works as expected and that you can display rectangular image and

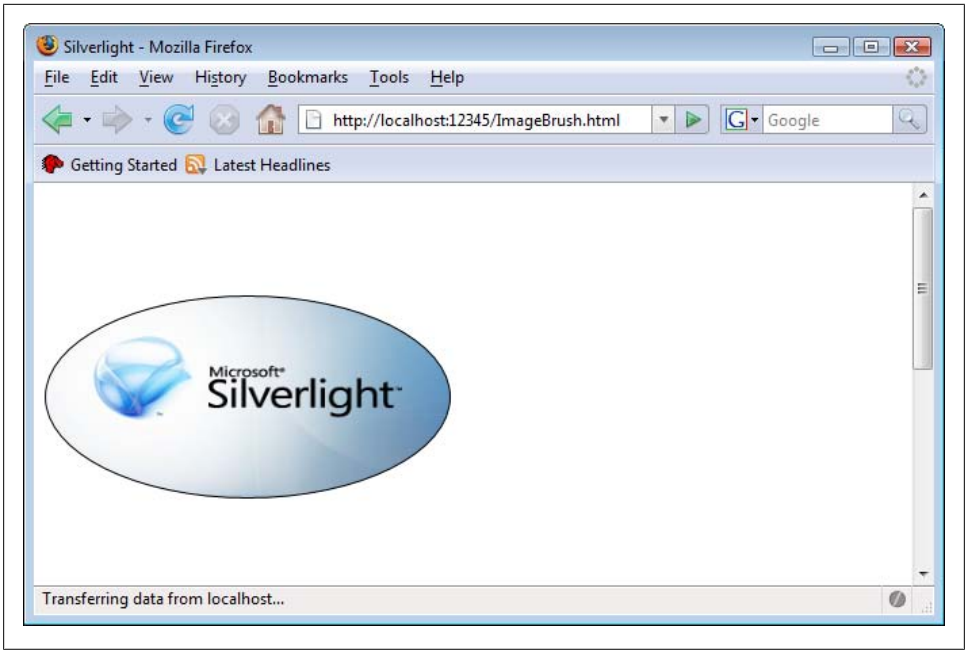


Figure 4-14. The ellipsis is filled with the image brush

video content and use other shapes. More information on using video in general can be found in Chapter 7.

Example 4-16. Using an image brush, the XAML file (ImageBrush.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="300" Height="300">
  <Ellipse Canvas.Top="75" Width="300" Height="150" Stroke="Black">
    <Ellipse.Fill>
      <ImageBrush ImageSource="silverlight.png" />
    </Ellipse.Fill>
  </Ellipse>
</Canvas>
```

An alternative approach to define the outline of an object is to use the `Clip` property and provide a `Geometry` object as its value, which will define the desired shape. Refer to the Silverlight SDK for more information (the topic is called "Silverlight Geometries Overview").

There are many shapes in XAML, almost too many to keep track of, but we covered the most important ones. And, honestly, usually the UI comes out of a design tool; as a programmer, you just have to add functionality and we'll start right away with that in the next chapter!

For Further Reading

XAML in a Nutshell (http://www.oreilly.com/catalog/xamlian/) by Lori A. MacVittie (O'Reilly)

A good introduction into XAML

CHAPTER 5

Interaction and Event Handling

Interactive Silverlight

The declarative means of XAML provide quite a number of possibilities, including creating all kinds of shapes (see Chapter 4), animating element (see Chapter 6), and playing audio and video data (see Chapter 7). However, you can only unleash the real power of Silverlight if you add a bit of JavaScript into the mix. JavaScript itself is a powerful language, but Silverlight also exposes a JavaScript API to developers. The third section of this book, especially Chapter 8, will cover the JavaScript access in greater detail. Although this book part focuses on declarative XAML, we cannot do without a certain amount of scripting.

This chapter explains the Silverlight event handling. Specifically, what events are and how to intercept and process them. Most of the event handling code will be in the "XAML code-behind" file (the *Filename.xaml.js* file). Technically, the JavaScript code could be placed in any JavaScript file, as long as it is referenced by a `<script>` tag in the main HTML file. However, it makes applications easier to comprehend and to develop if we stick JavaScript code that is triggered from XAML into the *.xaml.js* file and JavaScript code triggered by the HTML page in the *.html.js* file.

Most listings consist of at least three files—the HTML file, the XAML file, and one or more JavaScript files. Some of the information in those files is always very similar, for instance the code for the `createSilverlight()` function. In the next example we are adapting the reference to the embedded XAML file. Therefore, `createSilverlight()` will not be reprinted here. As long as there are no big surprises in the HTML file, it will not always be printed either. However, in the code archive for this book (see *http://www.oreilly.com/catalog/9780596516116*), you will have the complete set of samples.

When in doubt regarding the roles of the HTML, XAML, and JavaScript files, reviewChapter 2 where the basic concepts are explained.

Events and Event Handlers

A Silverlight event is something that occurs while a Silverlight application is running. This "something" could be a mouse click, a mouse movement, keyboard input, the application receiving or losing the focus, the application being fully loaded, or something else altogether. All events are tied to an object, so it makes a difference whether the mouse click occurs while the mouse pointer is, say, over a given rectangle or text block.

Silverlight currently supports more than two dozen events, and future releases may have even more. This chapter focuses on the most important ones, and provides the background knowledge you'll need to work with previously unknown events.

An event handler is a piece of code that is executed once an event occurs or is *fired*. The concept of both events and event handlers is very similar to the concept of JavaScript events in HTML.

Declarative Event Handlers

There are two ways of assigning an event handler to an event—using declarative means or using code. Let's start with the first option and use mouse events. Silverlight supports several mouse events, and one of them is `MouseLeftButtonDown`, which is when a user clicks down the left mouse button (this event occurs before the mouse button is released!).

In our first example, the "Hello World" sample file from Chapter 2 resurfaces. It contains three elements, the surrounding `<Canvas>`, a `<Rectangle>`, and a `<TextBlock>`. We are adding event handlers to these three elements. Adding an event handler is easy: assign an attribute that has the same name as the event (so, in this example, `LeftMouseButtonDown`, not `OnLeftMouseButtonDown` as you would do in JavaScript). The value of the attribute is a JavaScript function name. This function gets called once the event is fired. Example 5-1 shows the XAML file including the three event handlers:

Example 5-1. Using event handlers, the XAML file (MouseClick.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        MouseLeftButtonDown="mouseClick">
  <Rectangle Width="200" Height="75" Stroke="Orange" StrokeThickness="15"
             MouseLeftButtonDown="mouseClick" />
  <TextBlock FontFamily="Arial" FontSize="32" Canvas.Left="30" Canvas.Top="20"
             Foreground="Black" Text="Silverlight"
             MouseLeftButtonDown="mouseClick" />
</Canvas>
```

The JavaScript event handler function, residing in the *MouseClick.js.xaml* file, automatically receives two arguments, just as ASP.NET event handler functions do:

`sender`
: A reference to the object that has received the event and is therefore calling the event handler

`eventArgs`
: Information about the event; for example, you get the current mouse position when handling a mouse event

All that our simple event handling function does is to display which element fired the event. In order to do this, the string representation of the `sender` argument is used. For instance, if the `<TextBlock>` element fired the event, `sender.toString()` is `TextBlock`. Example 5-2 has the code in the "XAML code-behind", as I am referring to the *.xaml.js* files.

Example 5-2. Using event handlers, the XAML JavaScript file (MouseClick.xaml.js)

```
function mouseClick(sender, eventArgs) {
  alert('Ouch, says ' + sender.toString() + '!');
}
```

When you now click on the text block, two JavaScript pop-ups appear: One for the text block, and one for the canvas (see Figure 5-1). This is called event bubbling. Whenever an element receives an event, it then passes the event up to its next parent node (in our example, `<TextBlock>` passes the event to `<Canvas>`). If the parent node itself has a parent node as well, the event is passed further up. This mechanism, which is similar to the JavaScript event mechanism in Internet Explorer, is quite useful if you have a nested XAML structure and need to handle events for several objects. The setup in Example 5-1 is of course rather rare, usually you assign an event handler to just one element.

The `sender` property assigned to the event handler functions is also very convenient when you want to change the object that fired the event. A general rule of thumb is that every property (e.g., `Foreground` for a `<Rectangle>` object) is also exposed to JavaScript. Actually, Silverlight intercepts all JavaScript attempts to access properties and is transforms the request so that the correct Silverlight property is set to get. This is implemented in such a way that the access is case-insensitive; you could use both `Foreground` and `foreground`. The convention used in this book is to lower camelcase the properties: `Foreground` becomes `foreground`, `FontSize` becomes `fontSize`. Have a look at where two more mouse events are introduced: `MouseHover` (when the mouse pointer is over the display area of an element) and `MouseLeave` (when the mouse pointer leaves the display area of an element). Example 5-3 sets up a XAML file with these two events and also a specific foreground color.

Example 5-3. Changing event target properties, the XAML file (MouseHover.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Rectangle Width="200" Height="75" Stroke="Orange" StrokeThickness="15" />
  <TextBlock FontFamily="Arial" FontSize="32" Text="Silverlight"
             Canvas.Left="30" Canvas.Top="20"
```

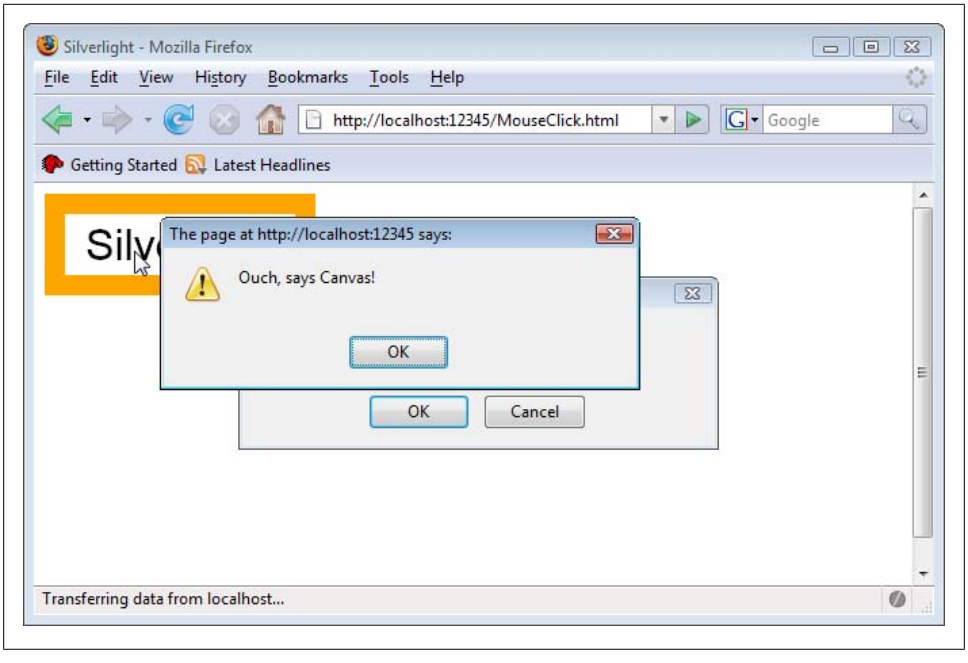


Figure 5-1. One, no, two pop-ups appear

```
                Foreground="LightGray" MouseEnter="high" MouseLeave="low"/>
</Canvas>
```

The "XAML code-behind" JavaScript file (see Example 5-4) accesses the `sender` argument and then just sets the `foreground` property. As an effect, the text is displayed in a light gray at first, but when the mouse hovers over it, it is colored with a full black. Once the mouse pointer leaves the display area of the text, it goes back to light gray again. Figure 5-2 shows both states of the text.

Example 5-4. Changing event target properties, the XAML JavaScript file (MouseHover.xaml.js)

```
function high(sender, eventArgs) {
  sender.Foreground = 'Black';
}
function low(sender, eventArgs) {
  sender.Foreground = 'LightGray';
}
```

Event Listeners

The second, and programmatic, approach to assign event handling code to an event is to use event listeners. You assign a piece of code that "listens" whether an event occurs. If it does, it is handled properly. The main advantage over declarative event handlers is that it is quite easy to remove an event listener. The disadvantage is that event listeners

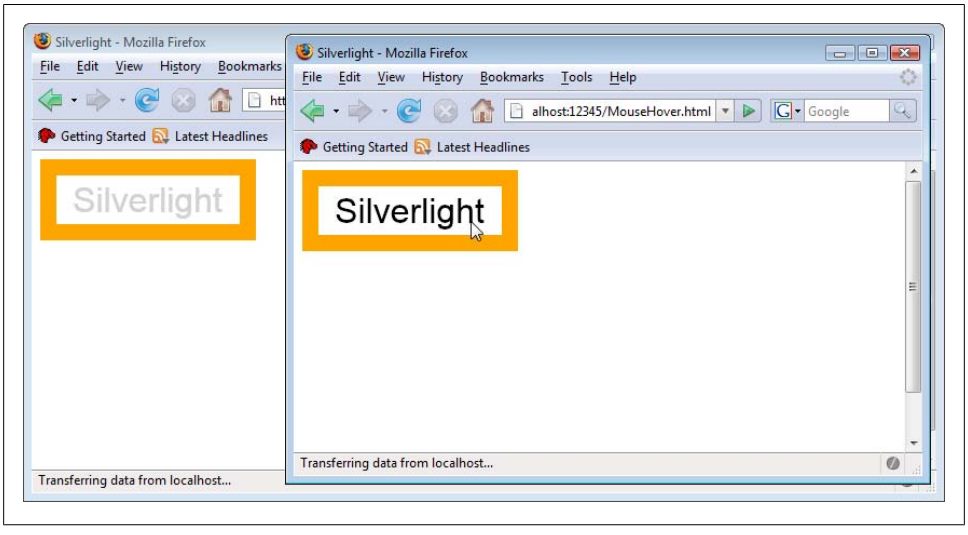


Figure 5-2. The initial (left) and hover (right) state of the text

are not as intuitive, especially for developers with a strong HTML background. But once you have seen it, getting it to work in your own applications is not hard at all.

Let's start with the XAML code. As you see, there is no attribute event handler for the click event. Well, there is one for another important event: `Loaded`. The `Loaded` event is fired once an element or, as in Example 5-5, the whole XAML file has been fully loaded. We use this event to set up the actual event listeners.

Example 5-5. Using event listeners, the XAML file (MouseClickListener.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Loaded="canvasLoaded">
  <Rectangle Width="200" Height="75" Stroke="Orange" StrokeThickness="15" />
  <TextBlock FontFamily="Arial" FontSize="32" Canvas.Left="30" Canvas.Top="20"
             Foreground="Black" Text="Silverlight"
             x:Name="ClickTarget" />
</Canvas>
```

Setting Up Event Listeners

Timing is everything. Let's have a look at the following JavaScript code to set up event listeners:

```
window.onload = function() {
  // set up event listeners here
}
```

The page's `load` event is fired once the complete HTML markup of the page is fully loaded. Of course, that does *not* mean that all referenced files, including the JavaScript libraries and the XAML file, have already been fully loaded. When testing this on your

local machine, it may not be an issue because the files were loaded from the fast hard drive. However, on the Internet there may be a significant lag, so you cannot rely on this approach. This is one of the reasons why the root element's `Loaded` event is a popular option for initializing a Silverlight application.

But now onto the event listeners themselves. First, you have to find the element you would like to attach the event listener to. As you may have noticed in Example 5-5, the `<TextBlock>` element received a name attribute (`x:Name`, to be exact). The Silverlight JavaScript API can find elements by their name, you just have to use the `findName()` method, which every element in a XAML file supports.

So we have to get access to an element within the XAML. Remember that event handler functions automatically receive a reference to the sender as their first argument. So we can use the sender and its `findName()` method to access the text block:

```
function canvasLoaded(sender, eventArgs) {
  var textblock = sender.findName('ClickTarget');
```

Once we have the text block (or any other element), we can assign an event handler to it. The method for this task is called `addEventListener()`, and it expects two arguments:

1. The name of the event
2. The event handler, either as a reference to a function, or as an anonymous function

```
textblock.addEventListener(
  'MouseLeftButtonDown',
  mouseClick);
```

All that's left to be implemented is the actual event handler, which outputs both the element that fired it and also its name. Example 5-6 shows the complete code for the JavaScript file, and Figure 5-3 depicts the output in the browser.

Example 5-6. Using event listeners, the XAML JavaScript file (MouseClickListener.xaml.js)

```
function canvasLoaded(sender, eventArgs) {
  var textblock = sender.findName('ClickTarget');
  textblock.addEventListener(
    'MouseLeftButtonDown',
    mouseClick);
}

function mouseClick(sender, eventArgs) {
  alert('Ouch, says ' + sender.toString() + ' "' + sender.name + '"!');
}
```

Removing event listeners will be covered in section "Mouse Position.

Mouse Events

Silverlight 1.0 supports these five mouse events, three of which you have already seen:

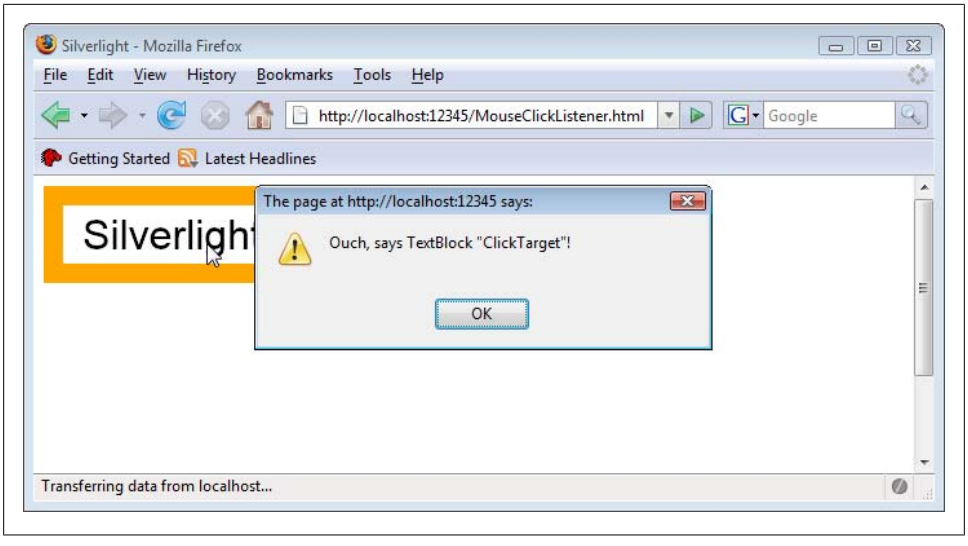


Figure 5-3. The event listener has been programmatically assigned

`MouseEnter`
: The mouse pointer entering the display area of an object

`MouseLeave`
: The mouse pointer leaving the display area of an object

`MouseMove`
: The mouse pointer moving

`MouseLeftButtonDown`
: The left mouse button being clicked down

`MouseLeftButtonUp`
: The left mouse button being clicked and released

The events themselves are self-explanatory, but the difference between `MouseLeftButtonDown` and `MouseLeftButtonUp` should probably be discussed. When a user clicks on an element, first `MouseLeftButtonDown` occurs, then `MouseLeftButtonUp`. So a mouse click is actually only complete when `MouseLeftButtonUp` has been fired. In the real world, the distinction only makes sense in one special case: the user hovers the mouse over an element, clicks the button, holds the button, and then moves the mouse away again. When you use `MouseLeftButtonUp`, it isn't fired over the target object, which is desirable in some scenarios (think buttons) and undesirable in other scenarios (think drag and drop).

The mouse event handling mechanism of the vector graphics format SVG, for instance, supports three mouse events: button pressed, button released, and a completed mouse click.

Mouse Position

When you capture a mouse event, you want to know where the mouse pointer currently is. The "where" question has previously been answered with "on this object." More specifically is the question "At which position?" This is where the second argument passed to event handler functions, `eventArgs`, comes into play. It provides access to this very information, by supporting the `getPosition()` method.

The `getPosition()` method supports an optional argument, which is any XML element. If this is set, `getPosition()` retrieves the *relative* position of the mouse to the given element. Otherwise, you get the `absolute` coordinates (i.e., if you do not provide an argument or if you provide `null`).

The return value of a `getPosition()` call is an object with the two properties `x` and `y`, which of course contain the horizontal and vertical positions of the mouse pointers. As always with the Web, the origin is in the top left corner.

Example 5-7 contains the XAML markup to track mouse movements. Note how the `<TextBlock>` property is used to display the location of the mouse pointer. Also, note that the main canvas' `Loaded` event executes a function called `canvasLoaded()`.

Example 5-7. Determining the Mouse Position, the XAML file (MousePosition.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Loaded="canvasLoaded">
  <Rectangle Width="200" Height="75" Stroke="Orange" StrokeThickness="15" />
  <TextBlock FontFamily="Arial" FontSize="28" Canvas.Left="25" Canvas.Top="25"
             Foreground="Black"
             Text="X: ??  Y: ??" x:Name="MousePosition" />
</Canvas>
```

The task of the JavaScript code is to determine and output the current mouse pointer position whenever the mouse is moving. The associated event name is `MouseMove`, and an event listener is ideal for this:

```
function canvasLoaded(sender, eventArgs) {
  sender.addEventListener(
    'MouseMove',
    // event listener reference or code
  );
}
```

All that's left to do is to write the event listener, so we will use an anonymous function here. The code determines the current mouse position using `getPosition()` and writes it into the text box. Refer to Example 5-8 for the complete JavaScript code, and to Figure 5-4 to see how this sample looks in the browser.

Example 5-8. Determining the mouse position, the XAML JavaScript file (MousePosition.xaml.js)

```
function canvasLoaded(sender, eventArgs) {
  sender.addEventListener(
    'MouseMove',
```

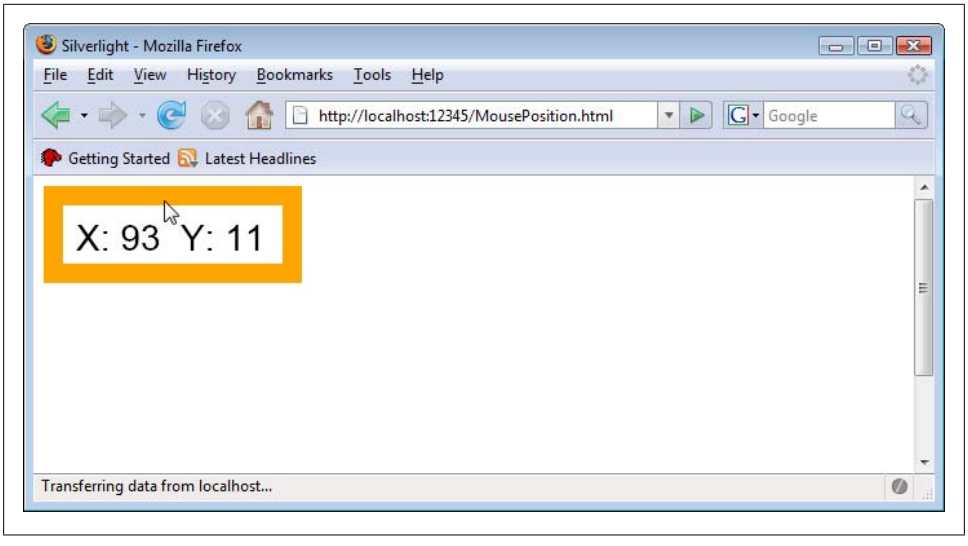


Figure 5-4. The current mouse position is displayed

```
        function(sender, eventArgs) {
          var x = eventArgs.getPosition(null).x;
          var y = eventArgs.getPosition(null).y;
          sender.findName('MousePosition').text =
            'X: ' + x + '  Y: ' + y;
        }
      );
    }
```

As mentioned before, you can also remove event listeners. To do this, you have to call the `removeEventListener()` method on the object the event listener has been attached to. The first argument is the event again, but the second argument may come as a surprise—it is a reference to the event listener (since you can attach more than one event listener to one single event). You generate this reference by saving the return value of the associated `addEventListener()` call.

To demonstrate this mechanism, we implement the hover effect again, but this time it can be enabled and disabled by clicking on the text. We start with the previous XAML markup from Example 5-7, but call the file *MousePositionToggle.xaml*. The JavaScript code, however, changes quite a bit. At first, we define two global variables. One will be used to save the attached event handler, and the other one is a Boolean value that tells the script whether we currently want to trace the mouse or not.

```
var traceMouse = false;
var handler = null;
```

Assigning the event handler to the canvas' `Loaded` event changes a bit, too: We assign a new function called `toggle()` to it:

```
function canvasLoaded(sender, eventArgs) {
  sender.addEventListener(
```

```
      'MouseLeftButtonDown',
      toggle);
}
```

The `toggle()` function first needs to check whether the mouse pointer coordinates are traced or not. If not, tracing must be enabled (since we want to toggle the behavior). We use the same code as before: whenever the mouse pointer moves, the new coordinates are displayed. Notice how the return value of the `addEventListener()` is saved in the global `handler` variable:

```
function toggle(sender, eventArgs) {
  if (!traceMouse) {
    handler = sender.addEventListener(
      'MouseMove',
      function(sender, eventArgs) {
        var x = eventArgs.getPosition(null).x;
        var y = eventArgs.getPosition(null).y;
        sender.findName('MousePosition').text =
          'X: ' + x + '  Y: ' + y;
      }
    );
```

If the mouse has been traced before (`traceMouse` equals to `true`), it must be deactivated; also, the event listener must be removed. A call to `removeEventListener()` takes care of that; remember that you have to use the `addEventListener()` return values as the second argument!

```
  } else {
    sender.removeEventListener('MouseMove', handler);
  }
```

Don't forget to toggle the `traceMode` variable: from `true` to `false`, from `false` to `true`:

```
  traceMouse = !traceMouse;
}
```

Example 5-9 contains the complete code of the XAML JavaScript file. If you run this example in the browser, you will need to click on the text to start seeing the mouse pointer coordinates. Again clicking on the text stops this.

Example 5-9. Adding and removing event listeners, the XAML JavaScript code (MousePositionToggle.xaml.js)

```
var traceMouse = false;
var handler = null;

function canvasLoaded(sender, eventArgs) {
  sender.addEventListener(
    'MouseLeftButtonDown',
    toggle);
}

function toggle(sender, eventArgs) {
  if (!traceMouse) {
    handler = sender.addEventListener(
```

```
        'MouseMove',
        function(sender, eventArgs) {
          var x = eventArgs.getPosition(null).x;
          var y = eventArgs.getPosition(null).y;
          sender.findName('MousePosition').text =
            'X: ' + x + '  Y: ' + y;
        }
      );
    } else {
      sender.removeEventListener('MouseMove', handler);
    }
    traceMouse = !traceMouse;
  }
```

Drag and Drop

One of the most difficult JavaScript effects is implementing drag and drop. Not only can it be hard to control individual elements on the page, but browser incompatibilities ultimately break the developer's neck. Silverlight does not come with built-in drag-and-drop support, but it is possible to implement this with rather little effort. If you plan it appropriately, the code will come together quickly.

Drag and drop always consists of three phases, which can be directly mapped on Silverlight mouse events:

`MouseLeftButtonDown`
: User clicks on draggable element and application enters drag mode.

`MouseMove`
: User moves the mouse, while the mouse button remains clicked. Selected object changes position according to the current position of mouse pointer.

`MouseLeftButtonUp`
: User releases the mouse button; application leaves drag mode.

Actually, this is about 50 percent of the solution. The other half comes from a different challenge. Let's assume that the draggable element is a 10 pixel by 10 pixel square. The user clicks somewhere on the square and drags it. Let's further assume the user releases the square at some position, for instance at (50,40) so the x-coordinate is 50 pixels and the y-coordinate is 40 pixels. Where should the JavaScript code now put the square? At (50,40)? This would place the top left corner of the square, so the position would only be correct if the user initially dragged the square by clicking exactly on the top left corner. This is rarely the case, of course.

So working with the absolute position is not a good solution. Instead, we will work with deltas: how far did the user move the mouse? In the first phase of drag and drop, JavaScript records the current position of the mouse pointer. Whenever the mouse is moved, the new position of the mouse pointer is retrieved. Based on these two values, JavaScript can calculate by how many pixels the mouse has been moved, for example 15 pixels to the right and 20 pixels to the bottom. These delta values can then be applied

to the actual object that shall be moved: it also must be moved 15 pixels to the right and 20 pixels to the bottom. (In reality, these pixel values are usually much smaller, since the `MouseMove` event is fired so often.)

This algorithm is the missing half of drag and drop. Writing the code is no big challenge any more. We start with the XAML code in Example 5-10. We once again have the surrounding rectangle and a black circle that will serve as the draggable object. Notice how the circle uses attributes to handle the three relevant mouse events.

Example 5-10. Drag and drop, the XAML file (DragDrop.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15" />
  <Ellipse Width="50" Height="50" Fill="Black" Canvas.Left="20" Canvas.Top="20"
    MouseLeftButtonDown="mouseInit"
    MouseLeftButtonUp="mouseRelease"
    MouseMove="mouseMove" />
</Canvas>
```

We can now start to implement the actual drag and drop. Before we do that, one word of caution. Remember the event bubbling mechanism? If the draggable object has processed a mouse event, it hands it over to its parent element. And even worse, other elements further down in the object chain could process and act upon this event, as well (this may also be called "event tunneling"). Therefore, other elements could work with those events too, which is usually undesirable in this scenario. So, Silverlight provides a `captureMouse()` method. If an object calls this method, all mouse events are directly routed to the object; other objects do not receive them any longer. When the left mouse button is pressed down (`MouseLeftButtonDown` event), the initial position of the draggable object is retained and the mouse capturing mode is activated. The (global) `moving` variable remembers whether the application is in drag and drop mode (`true`) or not (`false`).

```
function mouseInit(sender, eventArgs) {
  sender.captureMouse();
  lastX = eventArgs.getPosition(null).x;
  lastY = eventArgs.getPosition(null).y;
  moving = true;
}
```

If the mouse is moving, the position of the draggable object needs to be updated properly, according to the algorithm we designed at the beginning of this section. Remember, the current mouse position is determined, then the script code calculates the delta between the current and the last known position. The position of the draggable object is updated accordingly. Finally, the variables holding the last known coordinates are updated.

If you are updating the circle's position, you have to take care of one special issue: the property names you need to set are represented by the `Canvas.Left` and `Canvas.Top` attributes. However, JavaScript does not allow dots in property names, so `object`

name.Canvas.Left would not work. You can use JavaScript's array syntax instead: objectname['Canvas.Left']. Now all you have to remember is that the first argument of the event handler function is the object firing the event, in this case the element we want to position. Then the rest of the code is easy:

```
function mouseMove(sender, eventArgs) {
  if (moving) {
    var x = eventArgs.getPosition(null).x;
    var y = eventArgs.getPosition(null).y;
    sender['Canvas.Left'] += x - lastX;
    sender['Canvas.Top'] += y - lastY;
    lastX = x;
    lastY = y;
  }
}
```

You now see why we needed the moving variable. Mouse moving events happen all of the time, but you only want the circle to move if the user is dragging it!

The final step happens when the user releases the mouse button (LeftMouseButtonUp event). You can reset moving to false and unlock access to mouse events for other elements on the page by calling the releaseMouseCapture() method. We will implement one additional feature here. As you have seen, there is an orange rectangle in the XAML file. This serves as the barrier for the draggable element: The element must not touch or even leave this border.

We need to define a couple of variables that provide us with the minimum and maximum coordinates where the circle is allowed:

```
var minX = 15;
var maxX = 235;
var minY = 15;
var maxY = 85;
```

We will also save the position of the circle when the user starts a drag and drop operation (this code obviously belongs in the mouseInit() function):

```
startX = sender['Canvas.Left'];
startY = sender['Canvas.Top'];
```

Finally, when the user releases the mouse button, the current position is determined and checked against the valid coordinates. If the position is out of bounds, the draggable object is placed at the position it had at the beginning of the drag and drop operation:

```
var x = sender['Canvas.Left'];
var y = sender['Canvas.Top'];
if (x < minX || x > maxX || y < minY || y > maxY) {
  sender['Canvas.Left'] = startX;
  sender['Canvas.Top'] = startY;
}
```

Example 5-11 sums up the complete JavaScript code. Figure 5-5 shows the example in action: If the user releases the mouse button now, the circle would jump back to its original position.

Example 5-11. Drag and drop, the XAML JavaScript file (DragDrop.xaml.js)

```
var startX, startY, lastX, lastY;
var minX = 15;
var maxX = 235;
var minY = 15;
var maxY = 85;

var moving = false;

function mouseInit(sender, eventArgs) {
  sender.captureMouse();
  startX = sender['Canvas.Left'];
  startY = sender['Canvas.Top'];
  lastX = eventArgs.getPosition(null).x;
  lastY = eventArgs.getPosition(null).y;
  moving = true;
}

function mouseRelease(sender, eventArgs) {
  sender.releaseMouseCapture();
  moving = false;
  var x = sender['Canvas.Left'];
  var y = sender['Canvas.Top'];
  if (x < minX || x > maxX || y < minY || y > maxY) {
    sender['Canvas.Left'] = startX;
    sender['Canvas.Top'] = startY;
  }
}

function mouseMove(sender, eventArgs) {
  if (moving) {
    var x = eventArgs.getPosition(null).x;
    var y = eventArgs.getPosition(null).y;
    sender['Canvas.Left'] += x - lastX;
    sender['Canvas.Top'] += y - lastY;
    lastX = x;
    lastY = y;
  }
}
```

Keyboard Events

Using keyboard events on the Web is always a thankless task. It's not impossible, it's just very hard to get it right, and there are many factors that come into play. Different browsers have different approaches on how to determine keyboard input. Once a user types a key, you get the information that a key was pressed, but only the numeric value of that key. This works beautifully for easy characters like letters and numbers, but gets

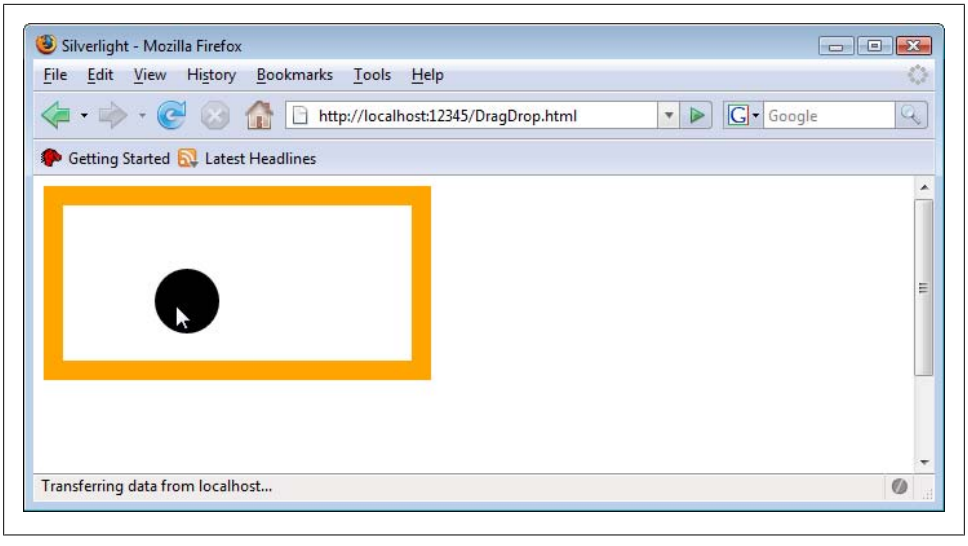


Figure 5-5. Drag and drop with Silverlight

harder when you have special keys, or even operating system specific keys. And if you have combination of keys, you are totally lost. For instance, Shift+9 usually types an opening parenthesis. However, with the keyboard layout I am using right now, Shift +9 creates a closing parenthesis.

Silverlight tries to amend this situation a little bit by providing two key codes in JavaScript:

- A nonplatform-specific key code
- A platform-specific key code

A general rule of thumb is that platform-specific key codes are close to ASCII codes (see *http://www.asciitable.com/* for a good list), whereas the nonplatform specific key codes work better for special characters.

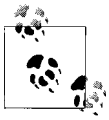
When capturing keyboard events, the `eventArgs` argument (the second one passed to the event handling function) provides the following properties:

`key`
: The non platform specific key code

`platformKey`
: The platform specific key code

`ctrl`
: Whether the Ctrl key (on Mac: the Apple key) has been pressed

`shift`
: Whether the Shift key has been pressed



At this time, Silverlight does not provide support for the Alt (Mac: Option) key. However, this key is rarely useful, since you need it for accessing browser menus (e.g., Alt + F for the *File* menu). Don't think that you just have to avoid the shortcut keys for your browser, because there are other browsers and other languages that use different shortcuts.

Another noteworthy point about Silverlight keyboard events is that you can only capture these events for the root element of your XAML file, the outer `<Canvas>`. We will now create a text input field that displays the text we entered, but need to attach the event to the `<Canvas>` element, as Example 5-12 shows. Such a text input field is not part of Silverlight 1.0, but is planned to be included in Silverlight 1.1.

Example 5-12. Capturing keyboard events, the XAML file (TextInput.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        KeyDown="keyPressed">
  <Rectangle Width="200" Height="75" Stroke="Orange" StrokeThickness="15" />
  <TextBlock FontFamily="Arial" FontSize="32" Canvas.Left="30" Canvas.Top="20"
             Foreground="Black" Text="" x:Name="InputBox" />
</Canvas>
```

In the JavaScript code we will need to determine which key has been pressed and then display it. Given that different keyboard layouts lead to different results, we make some very specific assumptions and:

- Target our code at the ASCII characters
- Allow only letters and numbers and the space character
- Allow only the backspace key to delete characters (this is more a feature than an assumption)

The ASCII table returns these key code ranges for the supported characters (except for the backspace key, which we will tackle upon in a bit):

- The space character has code 32
- The numbers run from 48 to 57
- The upper case letters run from 65 to 90, lower case letters run from 97 to 122

When pressing the *a* key, for instance, we get the key code of the capital *A*, 65. So if the Shift key has not been pressed, we need to add 32 to the key code to get the lower case letter's code (97 in this example). When the Space key is pressed, it does not matter whether Shift has been pressed or not. If a number key has been pressed, Shift must not be pressed (remember the Shift+9 example from the introduction of this section?).

So if one valid key or key combination has been pressed, we have a valid key code. The (built-in) JavaScript method `String.fromCharCode()` converts this information into the

"real" character, which allows the script to write the new character into the text field. To do this, the `findName()` method has once again to be used:

```
if (eventArgs.Ctrl) {
  return;
}
var keyCode = eventArgs.platformKeyCode;
if (keyCode == 32 || //space
   (keyCode >= 48 && keyCode <= 57 && !eventArgs.shift) || //numbers
   (keyCode >= 65 && keyCode <= 90)) { //letters
  if ((keyCode >= 65 && keyCode <= 90) && !eventArgs.shift) {
    keyCode += 32;
  }
  sender.findName('InputBox').text += String.fromCharCode(keyCode);
}
```

Onto the backspace key. Here the best option is to use the nonplatform-specific key code, which is easy to remember: `1`. If this key is pressed, the last character of the current input is removed. This is implemented with the JavaScript `substring()` method for strings:

```
if (eventArgs.key == 1) {
  var text = sender.findName('InputBox').text;
  sender.findName('InputBox').text = text.substring(0, text.length - 1);
}
```

Example 5-13 contains the complete JavaScript code. The application then uses letters, numbers, space characters and Backspace, and displays what you write (see Figure 5-6). If you want to complicate the code a little bit, you can add further characters and also experiment with the nonplatform-specific key codes to create a user experience that is as platform agnostic as possible.

Example 5-13. Capturing keyboard events, the XAML JavaScript file (TextInput.xaml.js)

```
function keyPressed(sender, eventArgs) {
  if (eventArgs.Ctrl) {
    return;
  }
  var keyCode = eventArgs.platformKeyCode;
  if (keyCode == 32 || //space
     (keyCode >= 48 && keyCode <= 57 && !eventArgs.shift) || //numbers
     (keyCode >= 65 && keyCode <= 90)) { //letters
    if ((keyCode >= 65 && keyCode <= 90) && !eventArgs.shift) {
      keyCode += 32;
    }
    sender.findName('InputBox').text += String.fromCharCode(keyCode);
  } else if (eventArgs.key == 1) {
    var text = sender.findName('InputBox').text;
    sender.findName('InputBox').text = text.substring(0, text.length - 1);
  }
}
```

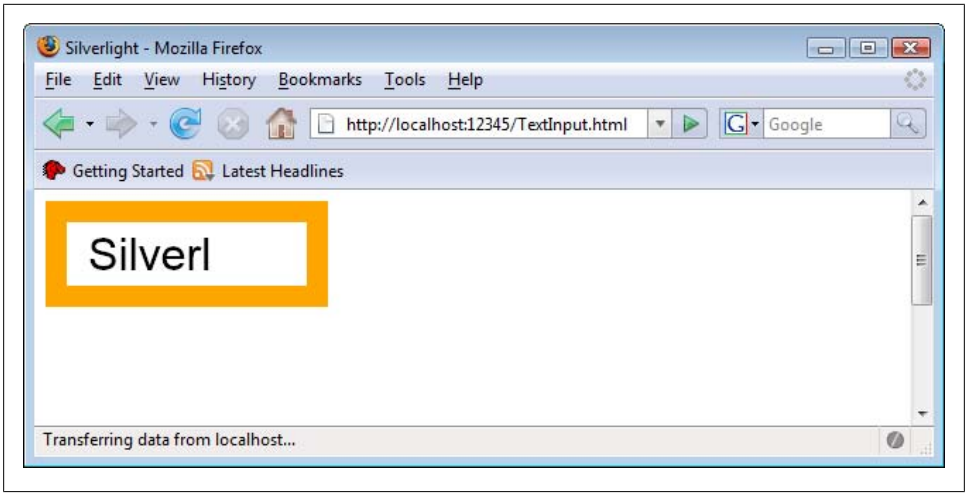
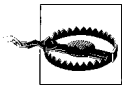


Figure 5-6. Keyboard input is displayed in the text field



When you run this example in the browser, you first need to click the Silverlight display area to activate the keyboard event handling. This is actually a security feature, otherwise plugins might capture keyboard input that a user was unwittingly entering while the browser had the focus. This is the same reason Silverlight's keyboard event handling does not work in full-screen mode. You may want to prompt users to click within the Silverlight content area prior to typing, either by specifically instructing them or by adding a button that needs to be clicked at the beginning.

Silverlight's event handling is not rocket science: You can use attributes to assign event handlers, or you write some code to attach (and maybe remove) event listeners. For keyboard and mouse events, the `eventArgs` event handler argument provides additional information about the event. There are other types of events too and the most interesting ones will be covered in the next chapters (e.g., media events in Chapter 7).

For Further Reading

http://www.asciitable.com/
: The ASCII table

http://msdn2.microsoft.com/en-us/library/ms645540.aspx
: Windows-specific key codes

http://developer.apple.com/documentation/carbon/reference/keyboardlayoutservices/KeyboardLayoutReference.pdf
: Mac-specific key codes

CHAPTER 6

Transformations and Animations

Transforming and Animating Content

Silverlight applications can be dynamic even if they do not use JavaScript. The content can be transformed and animated, both which will be covered in this chapter. The transformation is nothing that you can watch, it is more or less a calculation that takes effect when the Silverlight content is rendered by the plug-in. For example, you can rotate or skew elements. An animation, on the other hand, can really change the visual appearance of the applications: elements may move or change their color. As always, you will find many small and self-contained examples that showcase the most interesting and important Silverlight options.

Transformations

A transformation is technically only a change of one or more values. For instance, if an element is rotated, its position and the location of all the drawing points of the element change. If an element is moved to another position (that's a transformation, as well), basically the positions of all corners of the element change (if we leave fillings aside). Silverlight supports five transformations:

`TranslateTransform`
: Changes the position of an element

`ScaleTransform`
: Scales an element by multiplying its dimension horizontally and vertically

`SkewTransform`
: Skews an element by using a horizontal and a vertical angle

`RotateTransform`
: Rotates an element by using a angle

`MatrixTransform`
: Multiplies all points of an element by a given matrix,and uses the result as the new value

Let's start with the <TranslateTransform> element, which as just changes the position of an element, much like setting a related property like Canvas.Left or Canvas.Top would do.

To execute a transformation, you have to use the <*Element*.RenderTransform> element. The term needs to be replaced with the type of object you want to transform. For instance, if you want to transform a <Rectangle> object, use <Rectangle.RenderTransform>. If you want to transform a <TextBlock> object, use <TextBlock.RenderTransform>.

Within the <*Element*.RenderTransform> element, you need to put the transformation element; in this example it is <TranslateTransform>. This element expects two attributes, X and Y, denoting the new x and y coordinate of the associated element's left corner, as you can see here:

```
<TextBlock Text="...">
  <TextBlock.RenderTransform>
    <TranslateTransform X="10" Y="20" />
  </TextBlock.RenderTransform>
</TextBlock>
```

Example 6-1 starts with the Hello World example from Chapter 2 and translates both a rectangle and a text block. In the end result you will see both the original elements and the translated ones (Figure 6-1) .

Example 6-1. Translating elements, the XAML file (TranslateTransform.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15" />
  <TextBlock FontFamily="Arial" FontSize="56" Canvas.Left="25" Canvas.Top="40"
             Foreground="Black" Text="Silverlight" />

  <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15">
    <Rectangle.RenderTransform>
      <TranslateTransform X="350" Y="175" />
    </Rectangle.RenderTransform>
  </Rectangle>
  <TextBlock FontFamily="Arial" FontSize="56" Canvas.Left="25" Canvas.Top="40"
             Foreground="Black" Text="Silverlight">
    <TextBlock.RenderTransform>
      <TranslateTransform X="350" Y="175" />
    </TextBlock.RenderTransform>
  </TextBlock>
</Canvas>
```

Element scaling can be done using <ScaleTransform>. You can scale both horizontally (ScaleX attribute) and vertically (ScaleY attribute), and may also provide the center coordinates (CenterX and CenterY attributes). The scaling value is actually a factor. For instance, a scaling value of 2 doubles the horizontal or vertical size, wheres a factor of 0.5 halves it. Example 6-2 scales both ways, as Figure 6-2 shows.

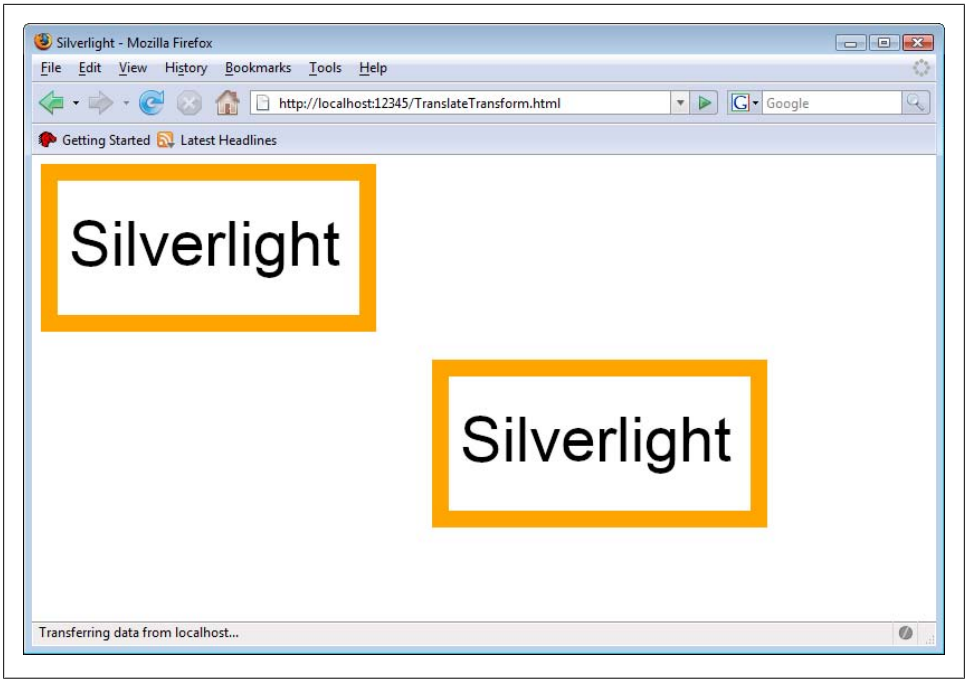


Figure 6-1. The original (left) and the translated (right) elements

Example 6-2. Scaling elements, the XAML file (ScaleTransform.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15" />
  <TextBlock FontFamily="Arial" FontSize="56" Canvas.Left="25" Canvas.Top="40"
             Foreground="Black" Text="Silverlight" />

  <Rectangle Canvas.Left="350" Canvas.Top="175"
             Width="300" Height="150" Stroke="Orange" StrokeThickness="15">
    <Rectangle.RenderTransform>
      <ScaleTransform ScaleX="1.5" ScaleY="0.5" />
    </Rectangle.RenderTransform>
  </Rectangle>
  <TextBlock FontFamily="Arial" FontSize="56"
             Canvas.Left="375" Canvas.Top="215" Foreground="Black"
             Text="Silverlight">
    <TextBlock.RenderTransform>
      <ScaleTransform ScaleX="1.5" ScaleY="0.5" />
    </TextBlock.RenderTransform>
  </TextBlock>
</Canvas>
```

The next transformation effect on our list is skewing, represented in Silverlight using the `<SkewTransform>` element. Skewing can be done using an horizontal angle (`AngleX`) and a vertical angle (`AngleY`). Again, you can provide the center of the transformation by setting the `CenterX` and `CenterY` attributes. The code in Example 6-3 uses a horizontal

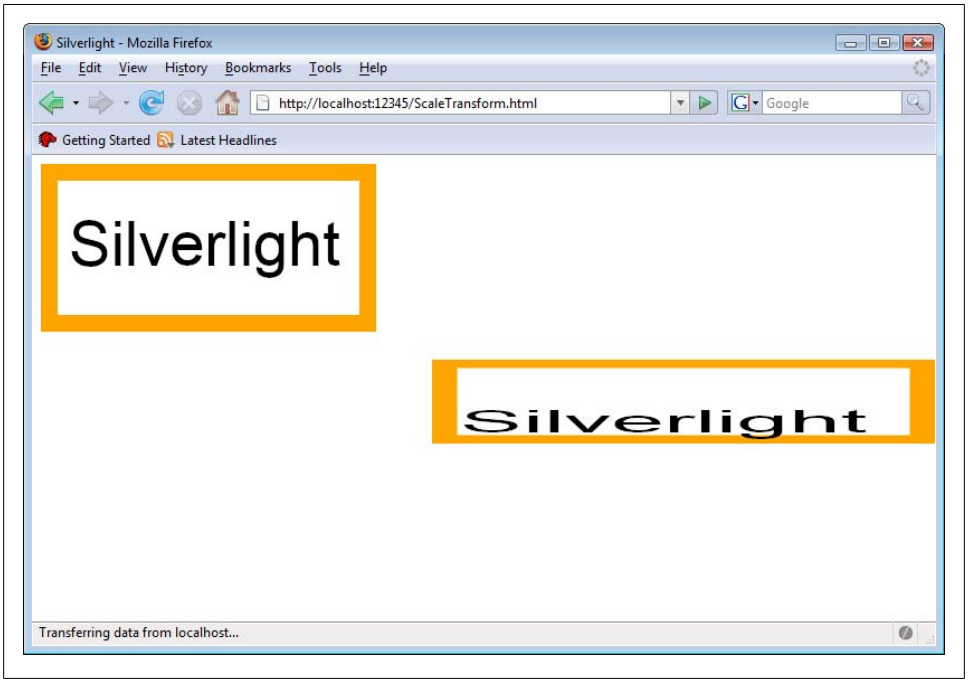


Figure 6-2. The original (left) and the scaled (right) elements

skewing of 45 degrees, and a vertical skewing of -30 degrees (which is the same as 330 degrees, because 360 degrees is a full circle). Figure 6-3 shows the result.

Example 6-3. Skewing elements, the XAML file (SkewTransform.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15" />
  <TextBlock FontFamily="Arial" FontSize="56" Canvas.Left="25" Canvas.Top="40"
             Foreground="Black" Text="Silverlight" />

  <Rectangle Canvas.Left="350" Canvas.Top="175"
             Width="300" Height="150" Stroke="Orange" StrokeThickness="15">
    <Rectangle.RenderTransform>
      <SkewTransform CenterX="50" CenterY="25"  AngleX="45" AngleY="-30" />
    </Rectangle.RenderTransform>
  </Rectangle>
  <TextBlock FontFamily="Arial" FontSize="56"
             Canvas.Left="375" Canvas.Top="215" Foreground="Black"
             Text="Silverlight">
    <TextBlock.RenderTransform>
      <SkewTransform CenterX="50" CenterY="25"  AngleX="45" AngleY="-30" />
    </TextBlock.RenderTransform>
  </TextBlock>
</Canvas>
```

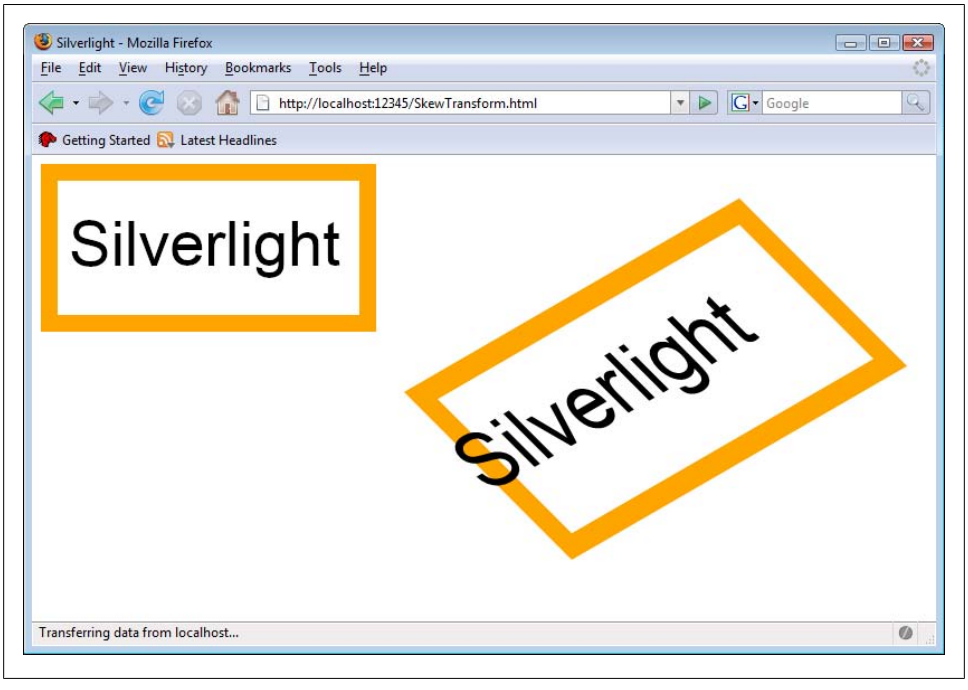


Figure 6-3. The original (left) and the skewed (right) elements

The final "simple" transformation is rotation, or `<RotateTransform>` in Silverlight. All you need to provide is the rotation angle (in degrees) in the `Angle` property. By default the element rotates around its top left corner, i.e., the relative coordinates (0,0). You can define this rotation point yourself by using the `CenterX` and `CenterY` attributes.

In Example 6-4, the `<Rectangle>` element is rotated by 45 degrees, around the point (150,50). To appropriately rotate the `<TextBlock>` element by the same degree, you need to take into account that the text block is translated 25 pixels to the right and 40 pixels to the bottom compared to the rectangle. This needs to be compensated in the rotation point, so we get (150-25,50-r0), or (125,10). As Figure 6-4 shows, the relative position of the text block within the rectangle remains intact this way.

Example 6-4. Rotating elements, the XAML file (RotateTransform.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15" />
  <TextBlock FontFamily="Arial" FontSize="56" Canvas.Left="25" Canvas.Top="40"
             Foreground="Black" Text="Silverlight" />

  <Rectangle Canvas.Left="350" Canvas.Top="175"
             Width="300" Height="150" Stroke="Orange" StrokeThickness="15">
    <Rectangle.RenderTransform>
      <RotateTransform Angle="-45" CenterX="150" CenterY="50" />
```

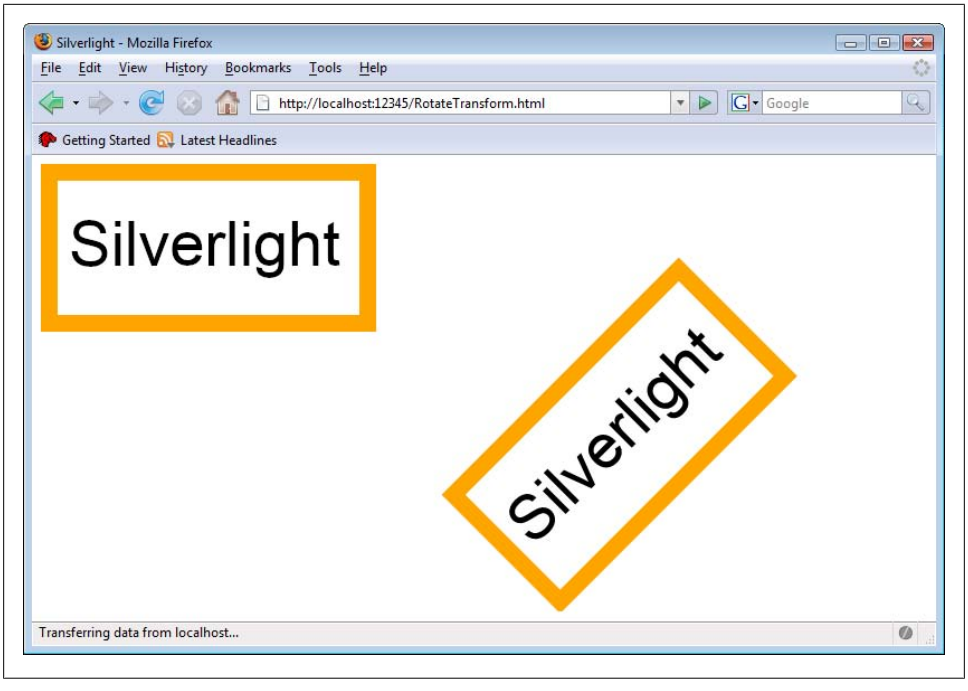


Figure 6-4. The original (left) and the rotated (right) elements

```
      </Rectangle.RenderTransform>
    </Rectangle>
    <TextBlock FontFamily="Arial" FontSize="56"
               Canvas.Left="375" Canvas.Top="215" Foreground="Black"
               Text="Silverlight">
      <TextBlock.RenderTransform>
        <RotateTransform Angle="-45" CenterX="125" CenterY="10" />
      </TextBlock.RenderTransform>
    </TextBlock>
  </Canvas>
```

To use more transformations at one time, you have to group them (the Silverlight object model and the XAML syntax do not allow multiple transform elements directly underneath `<Element.RenderTransform>`). But this limitation does not lead to an enormous extra effort, instead you just have to put an `<TransformGroup>` element within the `<Element.RenderTransform>` element and put your transformations in there. Example 6-5 shows such a transformation. As you can see in Figure 6-5, the code just translates the elements and skews them (you will notice that in contrast to previous listings, there are no `Canvas.Left` and `Canvas.Top` properties set for the transformed objects).

Example 6-5. Grouping transformations, the XAML file (TransformGroup.xaml)

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

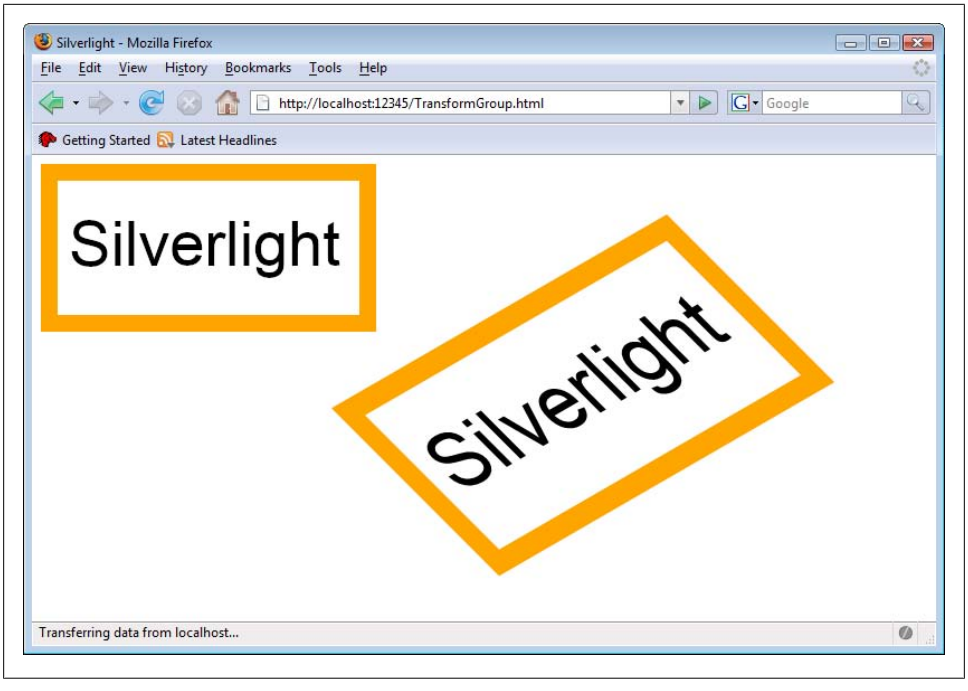


Figure 6-5. The original (left) and the transformed (right) elements

```
    <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15" />
    <TextBlock FontFamily="Arial" FontSize="56" Canvas.Left="25" Canvas.Top="40"
               Foreground="Black" Text="Silverlight" />

    <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15">
      <Rectangle.RenderTransform>
        <TransformGroup>
          <TranslateTransform X="350" Y="175" />
          <SkewTransform CenterX="425" CenterY="265" AngleX="45" AngleY="-30" />
        </TransformGroup>
      </Rectangle.RenderTransform>
    </Rectangle>
    <TextBlock FontFamily="Arial" FontSize="56" Foreground="Black"
               Text="Silverlight">
      <TextBlock.RenderTransform>
        <TransformGroup>
          <TranslateTransform X="375" Y="215" />
          <SkewTransform CenterX="425" CenterY="265" AngleX="45" AngleY="-30" />
        </TransformGroup>
      </TextBlock.RenderTransform>
    </TextBlock>
  </Canvas>
```

The complex calculations that are the basis for all Silverlight transformations can be further generalized. Every point (x,y) is multiplied with a 3x3 matrix to calculate its new coordinates. Since Silverlight only supports 2D, the third column of this matrix is

always (0,0,1), because we cannot use a z coordinate. However, the other two columns are used by the previously shown transformations. For instance, the first two values in the first two rows are used to scale and skew elements; the first two values in the third row of the matrix provide the value that an element is translated to. By setting all values at once, you can combine translating, skewing, and scaling transformations.

The math behind these calculations is beyond the scope of this book, but we will show you how this is done in markup. You need to use the `<MatrixTransform>` element; its `Matrix` property expects the first two values of each of the three matrix rows to be a comma separated list. Have a look at the code in Example 6-6 and then try to figure out what effect these values will have. The solution is shown in Figure 6-6.

Example 6-6. Transforming using a matrix, the XAML file (MatrixTransform.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15" />
  <TextBlock FontFamily="Arial" FontSize="56" Canvas.Left="25" Canvas.Top="40"
             Foreground="Black" Text="Silverlight" />

  <Rectangle Canvas.Left="300"
             Width="300" Height="150" Stroke="Orange" StrokeThickness="15">
    <Rectangle.RenderTransform>
      <MatrixTransform Matrix="1,1.5,1.25,1.25,1.5,1" />
    </Rectangle.RenderTransform>
  </Rectangle>
  <TextBlock FontFamily="Arial" FontSize="56" Canvas.Left="325"
             Foreground="Black" Text="Silverlight">
    <TextBlock.RenderTransform>
      <MatrixTransform Matrix="1,1.5,1.25,1.25,1.5,1" />
    </TextBlock.RenderTransform>
  </TextBlock>
</Canvas>
```

Transformations

UI elements aren't the only things that can be transformed. Brushes support transformations as well, using the `<NameOfBrush.Transform>` and `<NameOfBrush.RelativeTransform>` subelement (the latter element transforms using relative values). Also, geometry elements (used for shapes) support a `<NameOfGeometry.Transform>` subelement and can have transformations applied to them (see Chapter 4).

Animations

Animations are usually just a cheap visual effect, which is based on an element's properties being changed. So for instance, if an element moves from the top left to the bottom right corner of the canvas, its `Canvas.Left` and `Canvas.Top` properties are changed a few times a second. If an element fades in from out of nowhere, its `Opacity` value is animated,

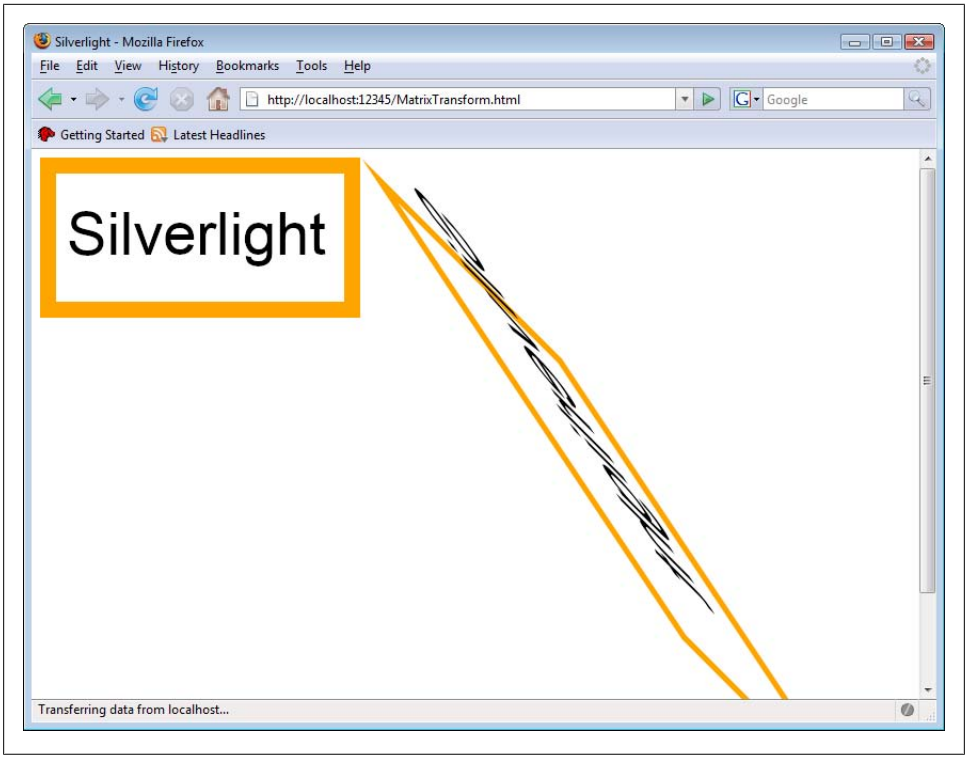


Figure 6-6. The original (left) and the transformed (right) elements

from 100 percent to 0 percent. So theoretically, you could rely solely on JavaScript and its access to Silverlight elements' properties to create animations. Of course, it is much more convenient to use the built-in animation support. Setting up an animation requires quite a number of steps, but the result can be very rewarding.

Setting Up an Animation

Creating an animation consists of several steps and a couple of lines of markup, so IntelliSense is really handy here. For example, to smoothly move an element to another position you need to animate the element. For reasons that will become clear in a minute, you should name that element.

```
<TextBlock FontFamily="Arial" FontSize="56" Canvas.Left="25" Canvas.Top="40"
           Foreground="Black" Text="Silverlight" x:Name="MyTextBlock">
...
</TextBlock>
```

Within this element, define a trigger using a `<TextBlock.Triggers>` element (if you were to animate a rectangle, you would use `<Rectangle.Triggers>`, and so on). An actual trigger (represented by `<EventTrigger>`) is activated when an event is fired. This event is provided in the `RoutedEvent` attribute of `<EventTrigger>`. Currently, Silverlight only

supports one event, *Element*.Loaded, where Element is the name of the object that contains the trigger (here it is TextBlock).

```
<TextBlock FontFamily="Arial" FontSize="56" Canvas.Left="25" Canvas.Top="40"
           Foreground="Black" Text="Silverlight" x:Name="MyTextBlock">
  <TextBlock.Triggers>
    <EventTrigger RoutedEvent="TextBlock.Loaded">
    ...
    </EventTrigger>
  </TextBlock.Triggers>
</TextBlock>
```

Within the event trigger, a *storyboard* is created. You need two elements <BeginStoryboard> and <Storyboard>. A storyboard is a set of one or more animations. You can try to compare what a storyboard does for animations to what the <TransformGroup> element does to transformations, which is group several of them together. An animation can consist of several individual animations, but more on that in a minute.

```
<TextBlock FontFamily="Arial" FontSize="56" Canvas.Left="25" Canvas.Top="40"
           Foreground="Black" Text="Silverlight" x:Name="MyTextBlock">
  <TextBlock.Triggers>
    <EventTrigger RoutedEvent="TextBlock.Loaded">
      <BeginStoryboard>
        <Storyboard>
        ...
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </TextBlock.Triggers>
</TextBlock>
```

Silverlight supports several animations, and they will all be covered in the next section. To complete the current example, let's introduce the most common animations: <DoubleAnimation>. This animation "animates" a value from a start value to an end value, for instance from 1 to 10. Every animation runs a certain amount of time. Within that interval, the associated animation value is gradually changed, from the start value to the end value. When the value goes from 1 to 10, these values might be 1, 1.1, 1.2, and so on until 10, depending on the duration of the animation. If the value that is animated is the x coordinate of the element to be animated, this creates the visual effect of the element smoothly moving from one point to another.

When using an animation, you will usually need several of these properties:

AutoReverse
: Reverses the animation if it has ended (i.e., moves the element back to where it started)

Duration
: The duration of an animation, using the syntax hh:mm:ss (hours, minutes, seconds)

From
: The start value for the animation

`To`
: The end value for the animation

`By`
: A relative value by how much to change the animation (alternative approach for using `To`)

`RepeatBehavior`
: What to do if the animation has ended (and optionally be reversed); you can provide a (total) duration, a number of times to repeat, or `Forever` if the animation should endlessly repeat

`Storyboard.TargetName`
: The name of the element that needs to be animated (therefore we needed to assign a name)

`Storyboard.TargetProperty`
: The property of the element that needs to be animated

The value of `Storyboard.TargetProperty` is the name of the property that receives the animated values. If the property includes a dot (such as in `Canvas.Left` or `Canvas.Top`), you need to enclose the complete property name in parentheses, e.g., `(Canvas.Left)`or`(Canvas.Top)`.

Adding a `<DoubleAnimation>` concludes the code, which is shown in Example 6-7. Both the rectangle and the text are moved 300 pixels to the right, using the default animation duration (here it is one second). Figure 6-7 has the output.

Example 6-7. Using <DoubleAnimation>, the XAML file (DoubleAnimation.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15"
             x:Name="MyRectangle">
    <Rectangle.Triggers>
      <EventTrigger RoutedEvent="Rectangle.Loaded">
        <BeginStoryboard>
          <Storyboard>
            <DoubleAnimation
              From="0" To="300.456"
              Storyboard.TargetName="MyRectangle"
              Storyboard.TargetProperty="(Canvas.Left)" />
          </Storyboard>
        </BeginStoryboard>
      </EventTrigger>
    </Rectangle.Triggers>
  </Rectangle>
  <TextBlock FontFamily="Arial" FontSize="56" Canvas.Left="25" Canvas.Top="40"
             Foreground="Black" Text="Silverlight" x:Name="MyTextBlock">
    <TextBlock.Triggers>
      <EventTrigger RoutedEvent="TextBlock.Loaded">
        <BeginStoryboard>
```

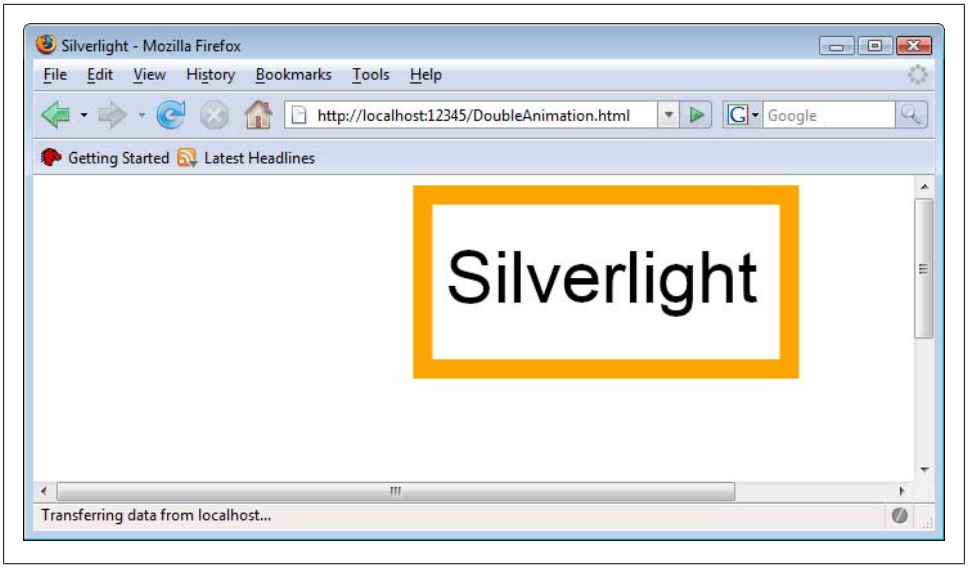


Figure 6-7. The elements are animated to the right (although it can't be seen in print)

```
            <Storyboard>
              <DoubleAnimation
                From="25" To="325.456"
                Storyboard.TargetName="MyTextBlock"
                Storyboard.TargetProperty="(Canvas.Left)" />
            </Storyboard>
          </BeginStoryboard>
        </EventTrigger>
      </TextBlock.Triggers>
    </TextBlock>
  </Canvas>
```

If you set up an animation that way, it will start immediately after the trigger has been activated. Silverlight allows you to change this behavior. Every animation also supports the `BeginTime` attribute, where you define the time (again using the `hh:mm:ss` syntax) when the animation starts. The code from Example 6-8 combines two `<DoubleAnimation>` elements: The first one moves the element to the right, the second one moves it to the bottom. The second animation starts after three seconds, which happens to be exactly the time when the first animation has ended. Figure 6-8 shows the second phase of the storyboard: the element is moving down.

Example 6-8. Combining animations and starting them later, the XAML file (DoubleAnimations.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15"
             x:Name="MyRectangle">
    <Rectangle.Triggers>
      <EventTrigger RoutedEvent="Rectangle.Loaded">
```

```
        <BeginStoryboard>
          <Storyboard>
            <DoubleAnimation
              From="0" To="300.456" Duration="0:0:3"
              Storyboard.TargetName="MyRectangle"
              Storyboard.TargetProperty="(Canvas.Left)" />
            <DoubleAnimation
              From="0" To="150" BeginTime="0:0:3" Duration="0:0:3"
              Storyboard.TargetName="MyRectangle"
              Storyboard.TargetProperty="(Canvas.Top)" />
          </Storyboard>
        </BeginStoryboard>
      </EventTrigger>
    </Rectangle.Triggers>
  </Rectangle>
  <TextBlock FontFamily="Arial" FontSize="56" Canvas.Left="25" Canvas.Top="40"
             Foreground="Black" Text="Silverlight" x:Name="MyTextBlock">
    <TextBlock.Triggers>
      <EventTrigger RoutedEvent="TextBlock.Loaded">
        <BeginStoryboard>
          <Storyboard>
            <DoubleAnimation
              From="25" To="325.456" Duration="0:0:3"
              Storyboard.TargetName="MyTextBlock"
              Storyboard.TargetProperty="(Canvas.Left)" />
            <DoubleAnimation
              From="40" To="190" BeginTime="0:0:3" Duration="0:0:3"
              Storyboard.TargetName="MyTextBlock"
              Storyboard.TargetProperty="(Canvas.Top)" />
          </Storyboard>
        </BeginStoryboard>
      </EventTrigger>
    </TextBlock.Triggers>
  </TextBlock>
</Canvas>
```

If you omit the `BeginTime` attribute in Example 6-8, both animations run at the same time, making the elements move diagonally.

Animation Types

Apart from `<DoubleAnimation>`, Silverlight also comes with support for two additional animations with a more specific purpose:

- `ColorAnimation` (animates a color value, e.g., `Orange`)
- `PointAnimation` (animates a point, e.g., `0,0`)

Animating a color works wherever you have a `Color` property. That means that you can *not* animate, say, the `Stroke` property. However, you can use a `SolidColorBrush` and animate the `Color` property there.

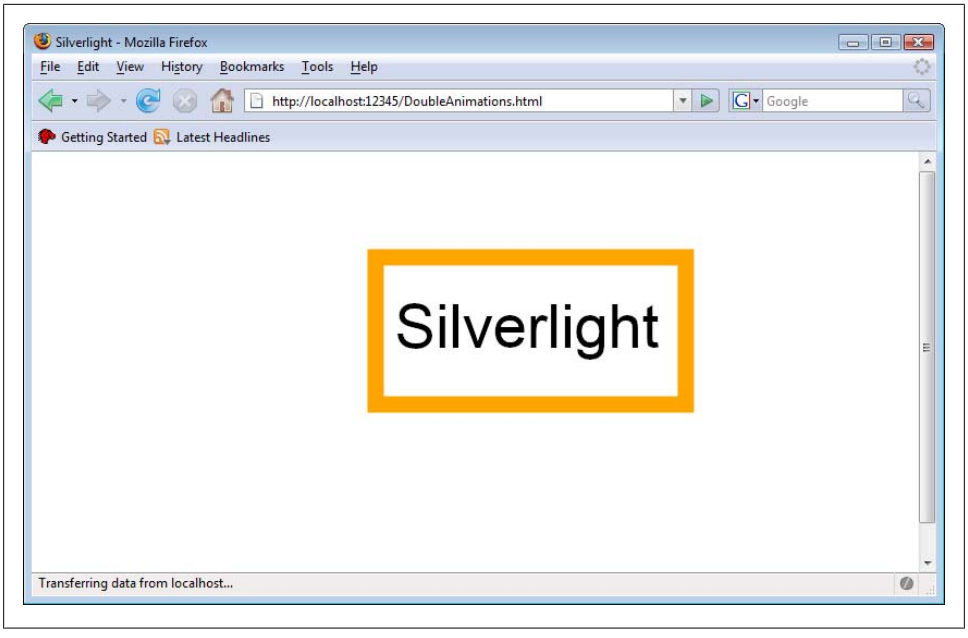


Figure 6-8. The elements first move to the right, then to the bottom

The rest is easy. Use the `<ColorAnimation>` element, set a `From` and a `To` color, and Silverlight takes care of the rest, as Example 6-9 shows (see Figure 6-9 for the browser output).

Example 6-9. Animating a color, the XAML file (ColorAnimation.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Rectangle Width="300" Height="150" StrokeThickness="15" x:Name="MyRectangle">
    <Rectangle.Triggers>
      <EventTrigger RoutedEvent="Rectangle.Loaded">
        <BeginStoryboard>
          <Storyboard>
            <ColorAnimation
              From="Green" To="Orange" Duration="0:0:5"
              Storyboard.TargetName="MyBrush"
              Storyboard.TargetProperty="Color" />
          </Storyboard>
        </BeginStoryboard>
      </EventTrigger>
    </Rectangle.Triggers>
    <Rectangle.Stroke>
      <SolidColorBrush x:Name="MyBrush" />
    </Rectangle.Stroke>
  </Rectangle>
  <TextBlock FontFamily="Arial" FontSize="56" Canvas.Left="25" Canvas.Top="40"
             Foreground="Black" Text="Silverlight" />
</Canvas>
```

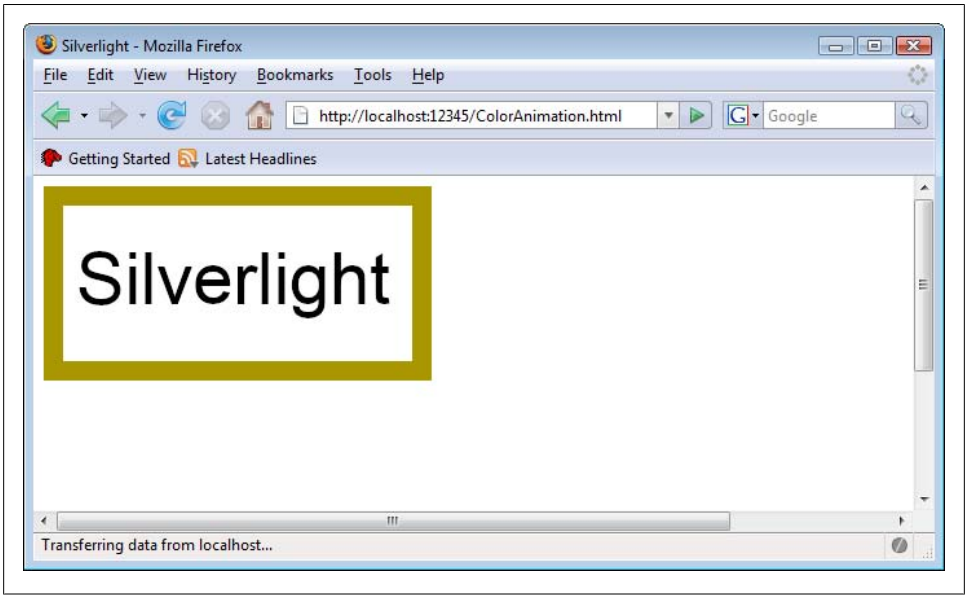


Figure 6-9. The stroke of the rectangle changes from green to orange

Internally, Silverlight is using the RGB values of the colors used and then animates these values. This is why you only get shades of grey when animating black (RGB values 0, 0, 0) to white (RGB values 255, 255, 255), since the colors "between" these two values have the same amount of red, green, and blue.

A point animation (`<PointAnimation>` in Silverlight's XAML) animates a point from a starting point to an end point (or by a certain distance in the coordinate system using `By`). To use it, you need to animate a property that requires a point as a value. One example for that is the `<LinearGradientBrush>`: its `StartPoint` and `EndPoint` attributes are both points. So, let's assume we have such a brush:

```
<LinearGradientBrush StartPoint="0,0" EndPoint="1,1" x:Name="MyGradient">
  <GradientStop Color="Red" Offset="0.0" />
  <GradientStop Color="Green" Offset="0.5" />
  <GradientStop Color="Blue" Offset="1.0" />
</LinearGradientBrush>
```

To animate this brush's `StartPoint` property, a `<PointAnimation>` element like the following would do:

```
<PointAnimation
  From="0,0" To="1,0" Duration="0:0:4"
  Storyboard.TargetName="MyGradient"
  Storyboard.TargetProperty="StartPoint" />
```

The code in Example 6-10 combines two animations: the start point of the gradient is moved from the top left to the top right corner and the end point of the gradient is moved from the bottom right to the bottom right corner. Figure 6-10 shows the Silverlight application in the browser during this animation: the gradient is moving.

Example 6-10. Animating a point, the XAML file (PointAnimation.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Rectangle Width="300" Height="150" StrokeThickness="15" x:Name="MyRectangle">
    <Rectangle.Triggers>
      <EventTrigger RoutedEvent="Rectangle.Loaded">
        <BeginStoryboard>
          <Storyboard>
            <PointAnimation
              From="0,0" To="1,0" Duration="0:0:4"
              Storyboard.TargetName="MyGradient"
              Storyboard.TargetProperty="StartPoint" />
            <PointAnimation
              From="1,1" To="0,1" Duration="0:0:4"
              Storyboard.TargetName="MyGradient"
              Storyboard.TargetProperty="EndPoint" />
          </Storyboard>
        </BeginStoryboard>
      </EventTrigger>
    </Rectangle.Triggers>
    <Rectangle.Stroke>
      <LinearGradientBrush StartPoint="0,0" EndPoint="1,1" x:Name="MyGradient">
        <GradientStop Color="Red" Offset="0.0" />
        <GradientStop Color="Green" Offset="0.5" />
        <GradientStop Color="Blue" Offset="1.0" />
      </LinearGradientBrush>
    </Rectangle.Stroke>
  </Rectangle>
  <TextBlock FontFamily="Arial" FontSize="56" Canvas.Left="25" Canvas.Top="40"
             Foreground="Black" Text="Silverlight" />
</Canvas>
```

Key Frame Animations

All of the animations so far were quite flexible in terms of the beginning and end time, but there was one severe restriction: every animation only took care of exactly one value that was animated. A bit more complex, but also a bit more flexible are so-called key frame animations. The term "key frame" is heavily used in Adobe Flash and identifies a frame during the course of a Flash movie where a certain state in the application must be reached (e.g., objects need to have specific positions). Silverlight only uses key frames as part of special animations, but the approach is comparable: when a key frame is reached, a certain object value must be met.

All three animation types we know so far (`<DoubleAnimation>`, `<ColorAnimation>`, and `<PointAnimation>`) can also be used with key frames. Then the names of the elements

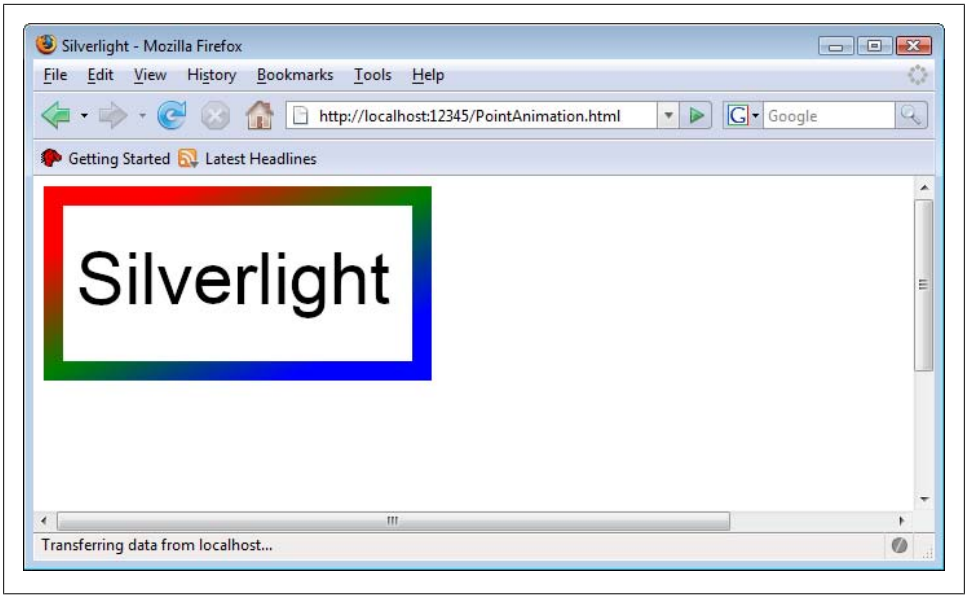


Figure 6-10. The gradient in the rectangle's stroke is changing

change (to `<DoubleAnimationUsingKeyFrames>`, `<ColorAnimationUsingKeyFrames>`, and `<PointAnimationUsingKeyFrames>`). Within each element, you provide the number of key frames. Every key frame needs at least two values, or attributes:

`KeyTime`
: The time when the key frame will come into effect

`Value`
: The value that needs to be reached at the given time

When you have one key frame after 2 seconds (providing a value of 10), and another one after 4 seconds (providing a value of 20), the value that you assign in these two frames will be animated between them. How this value is animated from 10 to 20 will be defined by the second key frame. There are different ways to interpolate the value, and the second key frame needs to confirm which of these methods is used. Three methods are currently supported by Silverlight:

Linear
: The value is linearly interpolated

Discrete
: There are no values between start and end value; when the next key frame is reached, the new value will be assigned

Spline
: The values will be animated along a cubic Bézier curve (the two control points are then provided in the `KeySpline` attribute

Every key frame can come in different flavors: different type of value that is animated (`Double`, `Color`, `Point`) and different type of interpolation method (`Linear`, `Discrete`, `Spline`). The name of the key frame element in XAML is the concatenation of interpolation type, value type, and `KeyFrame`. Therefore, a key frame that uses a spline interpolation of colors would be represented by the `<SplineColorKeyFrame>` element.

Apart from that, key frame animations just work as animations without key frames. So without further ado, have a look at Example 6-11 where the rectangle and the text block are moved around the canvas. The animations for the x and the y coordinate consist of four subanimations each. We are using a splined animation each time. Figure 6-11 shows the application in the animations.

Example 6-11. Animating using key frames, the XAML file (KeyFrameAnimation.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15"
             x:Name="MyRectangle">
    <Rectangle.Triggers>
      <EventTrigger RoutedEvent="Rectangle.Loaded">
        <BeginStoryboard>
          <Storyboard>
            <DoubleAnimationUsingKeyFrames
              Storyboard.TargetName="MyRectangle"
              Storyboard.TargetProperty="(Canvas.Left)"
              Duration="0:0:8">
              <SplineDoubleKeyFrame Value="300" KeyTime="0:0:2"
                                    KeySpline="0.25,0.75 0.75,0.25" />
              <SplineDoubleKeyFrame Value="100" KeyTime="0:0:4"
                                    KeySpline="0.75,0.25 0.25,0.75" />
              <SplineDoubleKeyFrame Value="50" KeyTime="0:0:6"
                                    KeySpline="0.25,0.75 0.75,0.25" />
              <SplineDoubleKeyFrame Value="200" KeyTime="0:0:8"
                                    KeySpline="0.75,0.25 0.25,0.75" />
            </DoubleAnimationUsingKeyFrames>
            <DoubleAnimationUsingKeyFrames
              Storyboard.TargetName="MyRectangle"
              Storyboard.TargetProperty="(Canvas.Top)"
              Duration="0:0:8">
              <SplineDoubleKeyFrame Value="50" KeyTime="0:0:2"
                                    KeySpline="0.25,0.75 0.75,0.25" />
              <SplineDoubleKeyFrame Value="250" KeyTime="0:0:4"
                                    KeySpline="0.75,0.25 0.25,0.75" />
              <SplineDoubleKeyFrame Value="50" KeyTime="0:0:6"
                                    KeySpline="0.25,0.75 0.75,0.25" />
              <SplineDoubleKeyFrame Value="100" KeyTime="0:0:8"
                                    KeySpline="0.75,0.25 0.25,0.75" />
            </DoubleAnimationUsingKeyFrames>
          </Storyboard>
        </BeginStoryboard>
      </EventTrigger>
    </Rectangle.Triggers>
  </Rectangle>
  <Canvas Canvas.Left="25" Canvas.Top="40">
```

```
    <TextBlock FontFamily="Arial" FontSize="56" Foreground="Black"
               Text="Silverlight" x:Name="MyTextBlock">
      <TextBlock.Triggers>
        <EventTrigger RoutedEvent="TextBlock.Loaded">
          <BeginStoryboard>
            <Storyboard>
              <DoubleAnimationUsingKeyFrames
                Storyboard.TargetName="MyTextBlock"
                Storyboard.TargetProperty="(Canvas.Left)"
                Duration="0:0:8">
                <SplineDoubleKeyFrame Value="300" KeyTime="0:0:2"
                                      KeySpline="0.25,0.75 0.75,0.25" />
                <SplineDoubleKeyFrame Value="100" KeyTime="0:0:4"
                                      KeySpline="0.75,0.25 0.25,0.75" />
                <SplineDoubleKeyFrame Value="50" KeyTime="0:0:6"
                                      KeySpline="0.25,0.75 0.75,0.25" />
                <SplineDoubleKeyFrame Value="200" KeyTime="0:0:8"
                                      KeySpline="0.75,0.25 0.25,0.75" />
              </DoubleAnimationUsingKeyFrames>
              <DoubleAnimationUsingKeyFrames
                Storyboard.TargetName="MyTextBlock"
                Storyboard.TargetProperty="(Canvas.Top)"
                Duration="0:0:8">
                <SplineDoubleKeyFrame Value="50" KeyTime="0:0:2"
                                      KeySpline="0.25,0.75 0.75,0.25" />
                <SplineDoubleKeyFrame Value="250" KeyTime="0:0:4"
                                      KeySpline="0.75,0.25 0.25,0.75" />
                <SplineDoubleKeyFrame Value="50" KeyTime="0:0:6"
                                      KeySpline="0.25,0.75 0.75,0.25" />
                <SplineDoubleKeyFrame Value="100" KeyTime="0:0:8"
                                      KeySpline="0.75,0.25 0.25,0.75" />
              </DoubleAnimationUsingKeyFrames>
            </Storyboard>
          </BeginStoryboard>
        </EventTrigger>
      </TextBlock.Triggers>
    </TextBlock>
  </Canvas>
</Canvas>
```

ScriptingAnimation

The final example in this chapter will show you how animations are exposed in the JavaScript code. This allows you to control animations from script code and will also provide a means to overcome the limitation of animations always starting when the XAML files were loaded. The example will start the animation when the mouse hovers over the elements on the canvas, pause it when the mouse leaves the canvas, and resume it again when the mouse pointer is back.

Since we need JavaScript code in the example, we create a "XAML code-behind" JavaScript file. Make sure that you include this file when implementing this example (see

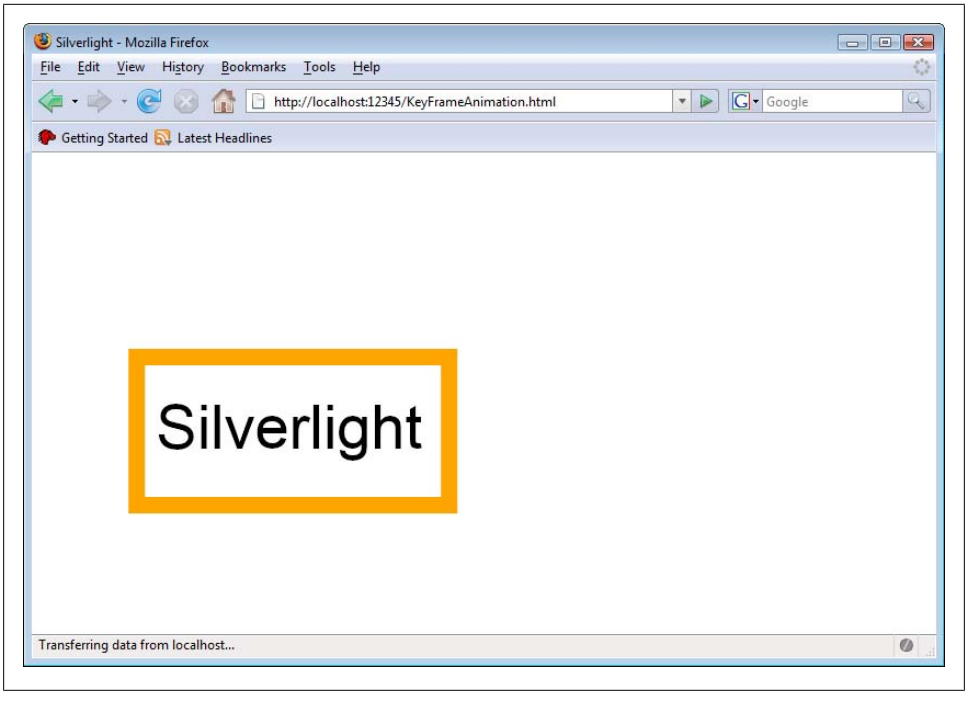


Figure 6-11. The elements move around the canvas, along Bézier curves

Example 6-12), otherwise the effects will not work (you will probably guess why I am including this here).

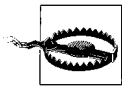
Example 6-12. Scripting animations, the HTML file (AnimationResources.html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Silverlight</title>

  <script type="text/javascript" src="Silverlight.js"></script>
  <script type="text/javascript" src="AnimationResources.html.js"></script>
  <script type="text/javascript" src="AnimationResources.xaml.js"></script>
</head>

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

To access an animation, you need to provide a name for the <Storyboard> element, because you can only control the playing state of storyboards, not individual animations. The code in this example introduces a new element, <Canvas.Resources>, which is not required for the example, but shows a nice way to separate animations from the elements that are animated. <Canvas.Resources> contains resources, which are elements that are referenced or used elsewhere in the Silverlight application. In this example, we just put two (named!) storyboards in there. Each storyboard contains a <DoubleAnimation> element that runs forever and we have also set AutoReverse to True.

```
<Canvas.Resources>
  <Storyboard x:Name="MyRectangleStoryboard">
    <DoubleAnimation
      From="0" To="300.456"
      Storyboard.TargetName="MyRectangle"
      Storyboard.TargetProperty="(Canvas.Left)"
      AutoReverse="True" RepeatBehavior="Forever"/>
  </Storyboard>
  <Storyboard x:Name="MyTextBlockStoryboard">
    <DoubleAnimation
      From="0" To="300.456"
      Storyboard.TargetName="MyTextBlock"
      Storyboard.TargetProperty="(Canvas.Left)"
      AutoReverse="True" RepeatBehavior="Forever"/>
  </Storyboard>
</Canvas.Resources>
```

As you can see, the two animations change the x coordinate of a rectangle (MyRectangle) and a text block (MyTextBlock).



Separating a storyboard from the animated element using <Canvas.Resources> can be taken one step further by omitting the Storyboard.TargetName property and setting dynamically from JavaScript. However, you can only do that if the animation has not yet started.

Example 6-13 shows the complete XAML markup. Note that there are two mouse event handlers in the <Canvas> element, but no triggers on the page.

Example 6-13. Scripting animations, the XAML file (AnimationResources.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        MouseEnter="beginAnimation" MouseLeave="pauseAnimation">
  <Canvas.Resources>
    <Storyboard x:Name="MyRectangleStoryboard">
      <DoubleAnimation
        From="0" To="300.456"
        Storyboard.TargetName="MyRectangle"
        Storyboard.TargetProperty="(Canvas.Left)"
        AutoReverse="True" RepeatBehavior="Forever"/>
    </Storyboard>
    <Storyboard x:Name="MyTextBlockStoryboard">
      <DoubleAnimation
```

```
          From="0" To="300.456"
          Storyboard.TargetName="MyTextBlock"
          Storyboard.TargetProperty="(Canvas.Left)"
          AutoReverse="True" RepeatBehavior="Forever"/>
      </Storyboard>
    </Canvas.Resources>
    <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15"
               x:Name="MyRectangle"/>
    <Canvas Canvas.Left="25" Canvas.Top="40">
      <TextBlock FontFamily="Arial" FontSize="56" Foreground="Black"
                 Text="Silverlight" x:Name="MyTextBlock"/>
    </Canvas>
  </Canvas>
```

Accessing such a storyboard is easy, just use the well-known `findName()` method introduced in Chapter 5 and provide the name of the `<Storyboard>` element. You can then control the animations using these five methods:

`begin()`
: Starts an animation at the beginning

`pauses()`
: Pauses an animation

`resume()`
: Resumes a paused animation

`stop()`
: Stops an animation

`seek(offset)`
: Jumps to a given position (using the `hh:mm:ss` syntax) in the animation

We now need to work on the event handlers. When the mouse leaves the canvas, we pause the animations using the `pause()` method. Here is the code:

```
function pauseAnimation(sender, eventArgs) {
  var storyboard1 = sender.findName('MyRectangleStoryboard');
  storyboard1.pause();
  var storyboard2 = sender.findName('MyTextBlockStoryboard');
  storyboard2.pause();
}
```

Starting the animations is a bit more complicated because the `begin()` method starts an animation (or a storyboard, to be exact) at the very beginning, but does not resume paused animations at their current position. On the other hand, the `resume()` method resumes a paused animation, but does not start a stopped one. Therefore, the JavaScript variable needs to remember whether the animation has already been started or not. Once an animation has been started, it will not be stopped (only eventually paused), since `RepeatBehavior` has been set to `Forever`. This allows us to implement the `MouseHover` event handler. Example 6-14 shows the complete code.

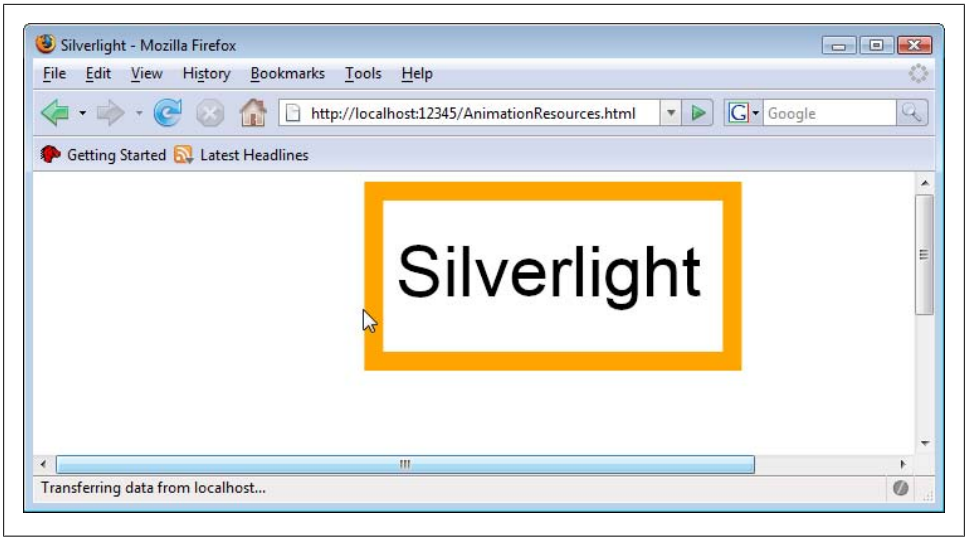


Figure 6-12. The mouse controls the animation

Example 6-14. Scripting animations, the XAML JavaScript file (AnimationResources.xaml.js)

```
var hasBegun = false;

function beginAnimation(sender, eventArgs) {
  var storyboard1 = sender.findName('MyRectangleStoryboard');
  var storyboard2 = sender.findName('MyTextBlockStoryboard');
  if (hasBegun) {
    storyboard1.resume();
    storyboard2.resume();
  } else {
    storyboard1.begin();
    storyboard2.begin();
    hasBegun = true;
  }
}

function pauseAnimation(sender, eventArgs) {
  var storyboard1 = sender.findName('MyRectangleStoryboard');
  storyboard1.pause();
  var storyboard2 = sender.findName('MyTextBlockStoryboard');
  storyboard2.pause();
}
```

Figure 6-12 shows the application in action (but you get the real experience when you try the code on your own). When the mouse hovers over the canvas, the animation starts. If you move the mouse pointer off the canvas, the animation stops, but if the mouse pointer returns, the animation continues at the same position it previously stopped.

Transformations and animations are quite different concepts, but both can achieve impressive effects with little efforts. With bigger projects, you will probably resort to

Microsoft Expression Blend 2 to get these effects up and running, but this chapter also showed you how to use scripting to provide additional functionality. And, by the way, you can also combine the two techniques presented in this chapter—transformations can be animated too.

For Further Reading

http://silverlight.net/quickstarts/silverlight10/animations.aspx
: Microsoft quickstart on Silverlight 1.0 animations

CHAPTER 7

Multimedia

Silverlight's Media Support

Adobe Flash has made a remarkable transition in the last years. The market penetration of the Flash player (the plug-in) has always been very high, but whenever a new player version came out, it took several months for it to reach a good-sized audience. However, in the last few months, this has sped up significantly. There may be many reasons for this, but one of the most compelling is that recent Flash versions have much better video support. Given that video sites like YouTube are extremely popular at the moment, they prompt users to install the latest player to see the content.

Multimedia support is a key feature of many browser plugins, and Silverlight clearly does not want to disappoint its users here. For obvious reasons, the supported media formats are biased toward Microsoft's offerings. Silverlight supports Windows Media Audio (WMA) and Windows Media Video (WMV) files, version 7 through 9. Also WMVA and WMVC1, two rather new video formats by Microsoft, are supported. The only external format Silverlight can process is the very popular MP3 audio format.

There is a reason for this bias, however. The plug-in plays content in these formats without the help of any other software or player. So, it is not necessary to have an MP3 player or even Windows Media Player to play supported multimedia content in Silverlight. This applies to both the Windows and Mac platforms.

Silverlight also supports streaming, either in the form of Windows Media Server streaming data, or ASX files. Note, however, that the stream support has some limitations: the content may not be paused and not all ASX features are supported. Refer to the Silverlight SDK for a list of restrictions when using streamed multimedia data.

Preparing Multimedia Data

Ideally, you already have your audio or video data in the correct format and can directly embed it into your Silverlight application (see "Embedding Multimedia for details). However, there is usually at least one step left to do–converting the audio data. For

example, the audio data could be in the wrong format, the video data might be too big for reasonable web playback, or you would like to add some markers to a video presentation. Some helpful tools can be used to get your multimedia data into the right format.

Converting Data

Starting with Windows XP, Microsoft began bundling a simple but reasonable video editing tool: the *Windows Movie Maker*. If you are using Windows XP and don't see it in your Start menu, you might want to visit Windows Update (menu entry in Internet Explorer's *Tools* menu) and install it from there. Windows Vista users will find the Windows Movie Maker in their Start menu. However, there is a catch; if you have an outdated video card, its hardware acceleration may not be good enough for Windows Movie Maker (see Figure 7-1). Also, some editions of Windows don't come with Windows Movie Maker. If this is the case, there is one more option: an older Windows Movie Maker version, 2.6, has been made available for Windows Vista. You can download it from *http://www.microsoft.com/downloads/details.aspx?FamilyID=d6ba5972-328e-4df7-8f9d-068fc0f80cfc&DisplayLang=en*. Note that Windows Movie Maker needs some libraries installed by Windows Media Player. Some Windows editions (the ones whose names are ending in "N", e.g., Vista Business N) do not come with Windows Media Player, but you can download it, too: *http://www.microsoft.com/downloads/details.aspx?FamilyID=1d224714-e238-4e45-8668-5166114010ca&DisplayLang=en*. Do not let the name of the download page confuse you: Although it says that you get "Windows Media Player 11 for Windows XP," it also runs under Windows Vista and restores the Windows Media components there, as you can see in Figure 7-2.

Make sure that you visit Windows Update after installing Windows Media Player, since there have been some security updates for this product recently.

You can import a number of formats into Windows Movie Maker, including AVI. Within the software, you can also cut the video and add special effects, including rotating effects (see Figure 7-3). For instance, the sample video used in this chapter was originally recorded in portrait mode, but cameras usually default to landscape mode. Windows Movie Maker provided an easy way to get the video into the right orientation.

After editing the video, you can export it using *File/Save Movie File*. You can tune the output format quite a bit (see Figure 7-4). If you optimize it for web play, you have a lower quality but the file size can get drastically smaller. If you want to retain reasonable video and audio quality, the trade off may be a big file size. Experiment a bit with the settings to find a good compromise between size and quality.

Another video editing option is a tool from the Microsoft Expression offerings (Chapter 3 introduced other tools from that product line): the Microsoft Expression Media

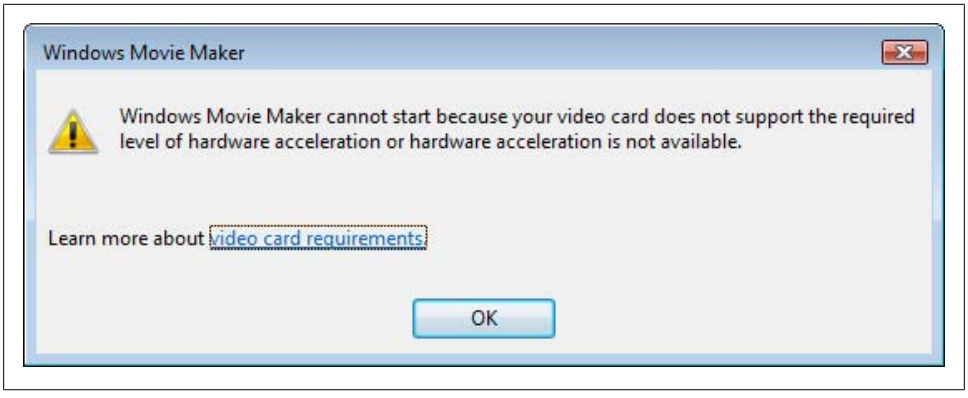


Figure 7-1. The graphics card is not good enough for Windows Movie Maker

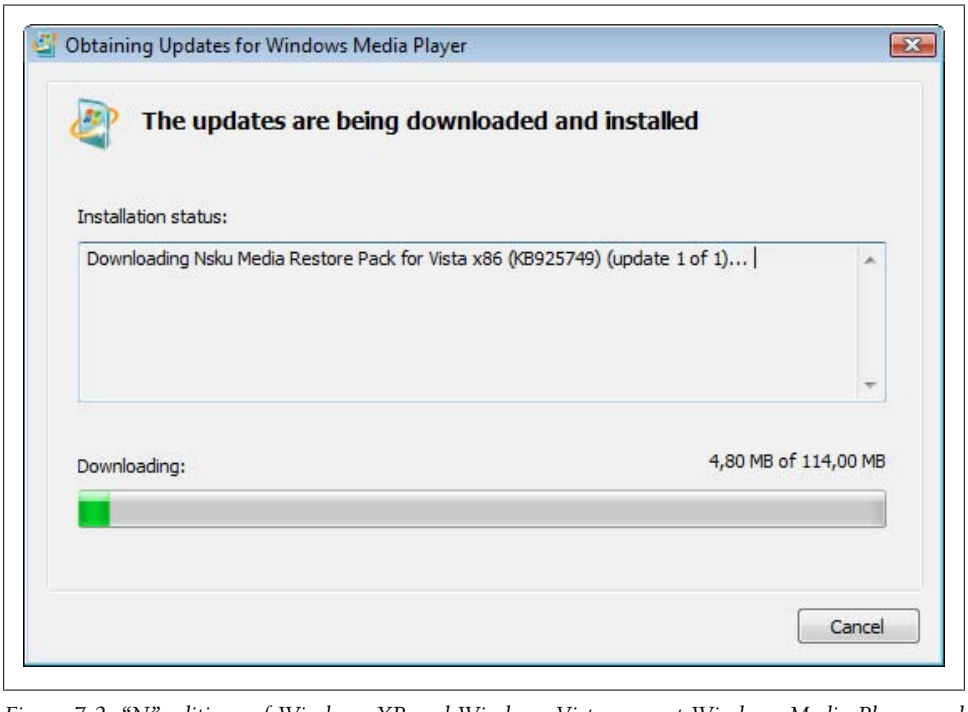


Figure 7-2. "N" editions of Windows XP and Windows Vista can get Windows Media Player and Windows Movie Maker, too

Encoder (see Figure 7-5). As of time of writing, a time-limited trial version is available for download from *http://www.microsoft.com/downloads/details.aspx?FamilyID=ba187636-abb6-4e55-9706-5bd346e39ea9&DisplayLang=en* (the full version can be purchased). The Expression Media Encoder installs special profiles for generating Silverlight content.

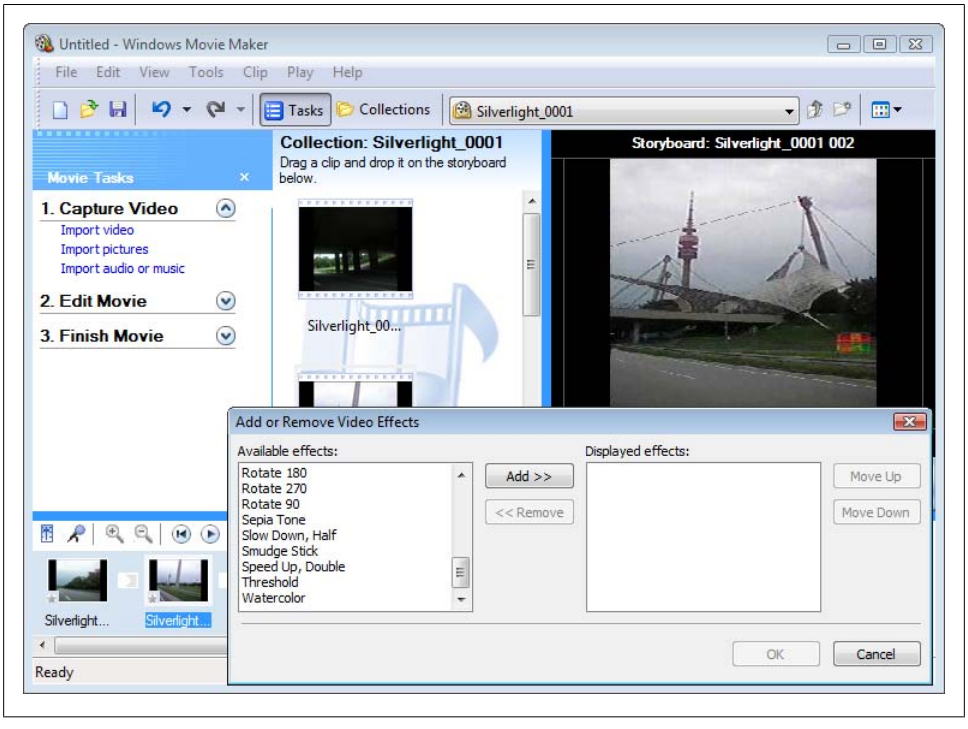


Figure 7-3. Windows Movie Maker

A third option, only available for Windows versions 2000 and XP, is the Windows Media Encoder 9 series, available at *http://www.microsoft.com/downloads/details.aspx?FamilyID=5691ba02-e496-465a-bba9-b2f1182cdf24&DisplayLang=en*. Figure 7-6 shows the software in action.

Adding Markers

An advanced multimedia feature that can be really convenient especially with video data is the marker support. This lets you mark special points within a media file. You can compare this to the chapters of a movie on DVD, where you can jump between chapters to reach a certain point in the movie.

The same things are valid for markers. You can define markers, either within the media file (covered in this section) or also, temporarily, within the Silverlight application (covered in section "Working with Markers). Silverlight provides a JavaScript API that can access markers and also determine when a marker has been reached.

The *Windows Media File Editor* component of the Windows Media Encoder allows markers to be inserted into Microsoft's video formats. Figure 7-7 shows what this looks like. Just navigate to certain positions in the file, provide a name, and you have a marker.

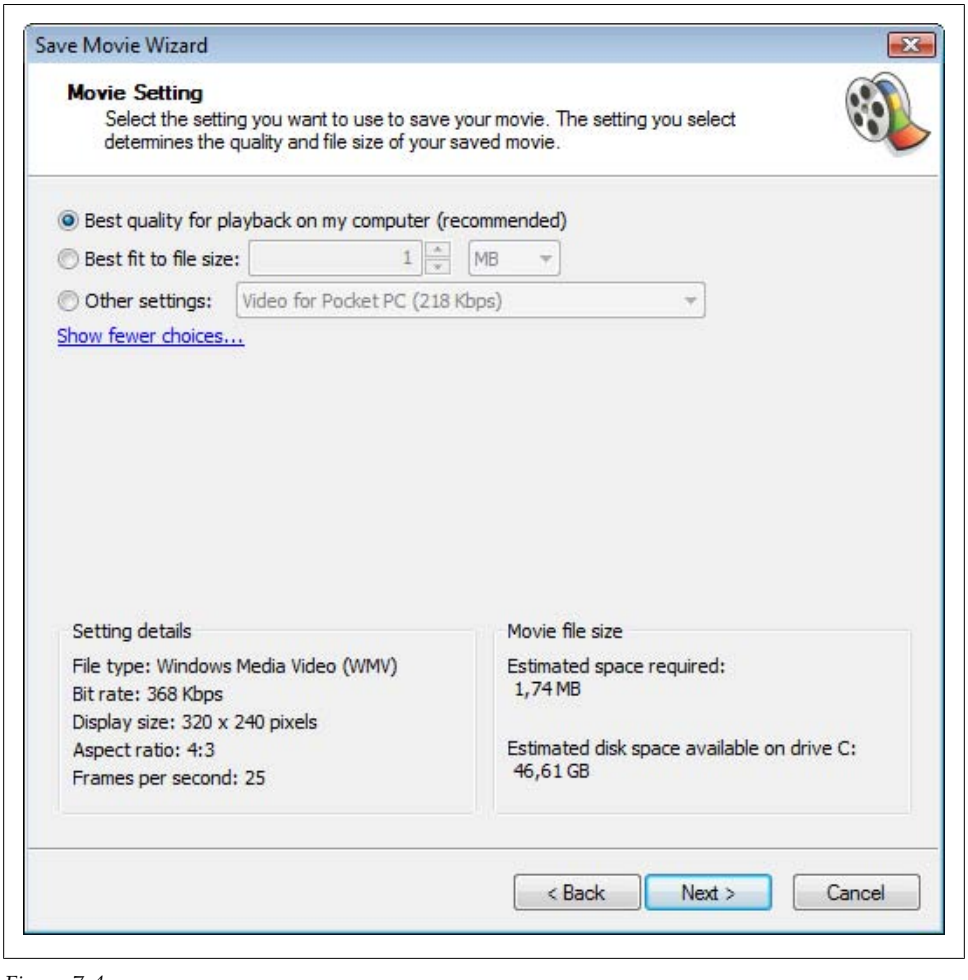


Figure 7-4.

Microsoft Expression Media Encoder also allows setting markers in the UI. The marker editor resides by default in the top right corner and is displayed in Figure 7-8.

Streaming Video

When you convert media data into one of the supported formats, the whole file will then be loaded from the Silverlight plug-in and play. A much better option of course is to use streaming. With streaming, the file resides at a (usually fast) remote server, the amount of data transferred can be adapted to the connection speed of the current user, and it is impossible to directly download the media file. Various streaming options exist, but for Silverlight users, the Windows Media Server is a good choice. Microsoft

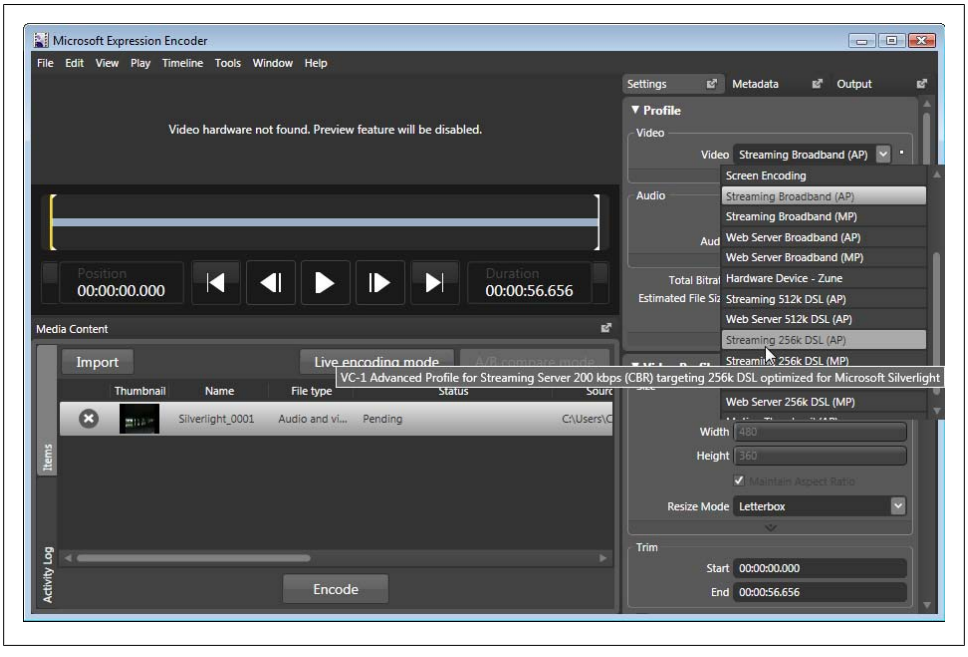


Figure 7-5. Microsoft Expression Media Encoder

has recently started offering more or less free Silverlight streaming. "More or less free" means that there are some restrictions, but for many scenarios, they do not hurt the application. Currently, the maximum speed for streaming is 700 kbps, and the streamed files may be add up to 4 GB of space. The streaming service is currently in alpha and later versions will add additional restrictions, but plans include 1 million minutes of streaming per month or unlimited streaming if you agree to have advertisements shown.

The streaming service's home page, *http://silverlight.live.com/*, contains up-to-date information about the streaming service itself and also on the current restrictions.

There are two ways to use the streaming service. You can either host your complete Silverlight application on the streaming server, or just put your video file up on the server and reference it in your locally hosted Silverlight application. At the time of this writing, the service and API are frequently changing, so we will only briefly discuss it.

Figure 7-9 shows the streaming site, and Figure 7-10 shows how you can upload your own files and applications to the service. More information can be found in the web-based Silverlight streaming SDK at *http://dev.live.com/silverlight/*.

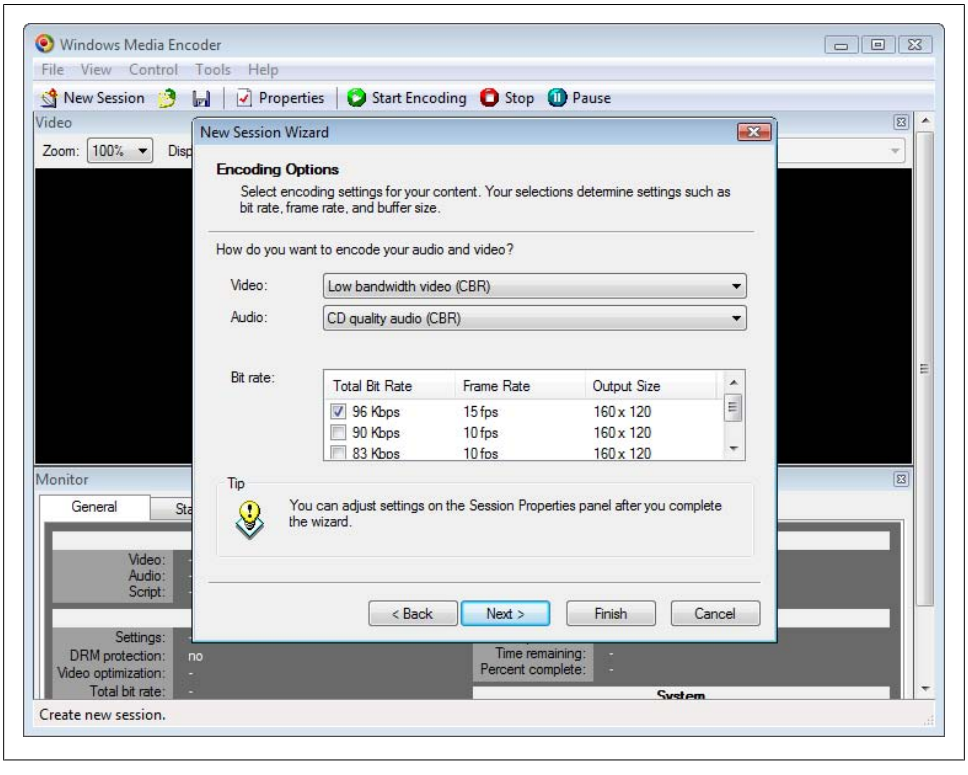


Figure 7-6. Windows Media Encoder

If you are using Microsoft Expression Media Encoder, the application already comes with suitable Silverlight profiles for exporting content. However, when you want to use the Windows Media Encoder, you can find Silverlight profiles at *http://dev.live.com/silverlight/downloads/profiles.zip*.

MediaElement

Regardless whether you want to use video or audio data, you only need to know about one element: `<MediaElement>`. It takes care of playing the media content, as long as the format is supported.

Embedding Multimedia

`<MediaElement>` supports a number of attributes, some of them are general, some of them are media-specific. For example, the `NaturalVideoWidth` and `NaturalVideo`

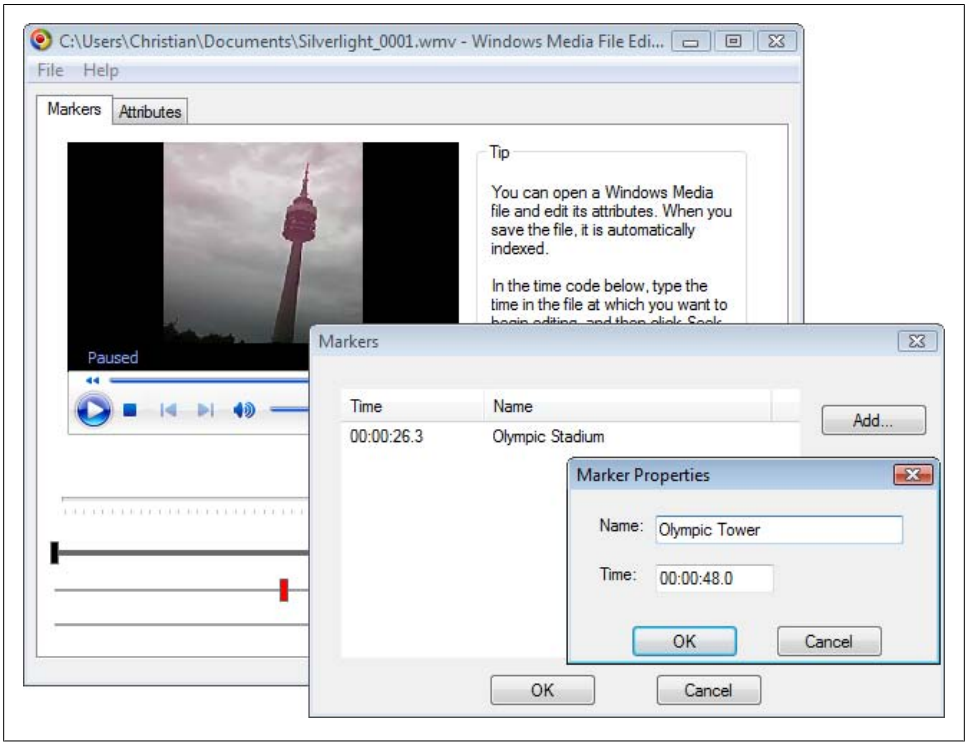


Figure 7-7. Setting markers in Windows Media File Editor

`Height` properties/attributes determine the size of a video. Obviously, this does not make sense for audio content because these two properties have the value `0`.

The most important attribute is `Source`. It provides the URL of the audio or video data to play. Since audio data is just a special case of video data (no visual output), we will focus on video in this chapter. Example 7-1 shows the most simple Silverlight video player imaginable. Figure 7-11 shows the output in the browser.

Example 7-1. A simple video player (Player.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Canvas.Left="25" Canvas.Top="25">
  <MediaElement Source="video.wmv"/>
</Canvas>
```

What's missing in Figure 7-11? The UI for using the player. This is not shipped as part of JavaScript; however, Silverlight provides a powerful JavaScript API for controlling multimedia content which will be discussed in "Controlling Multimedia.

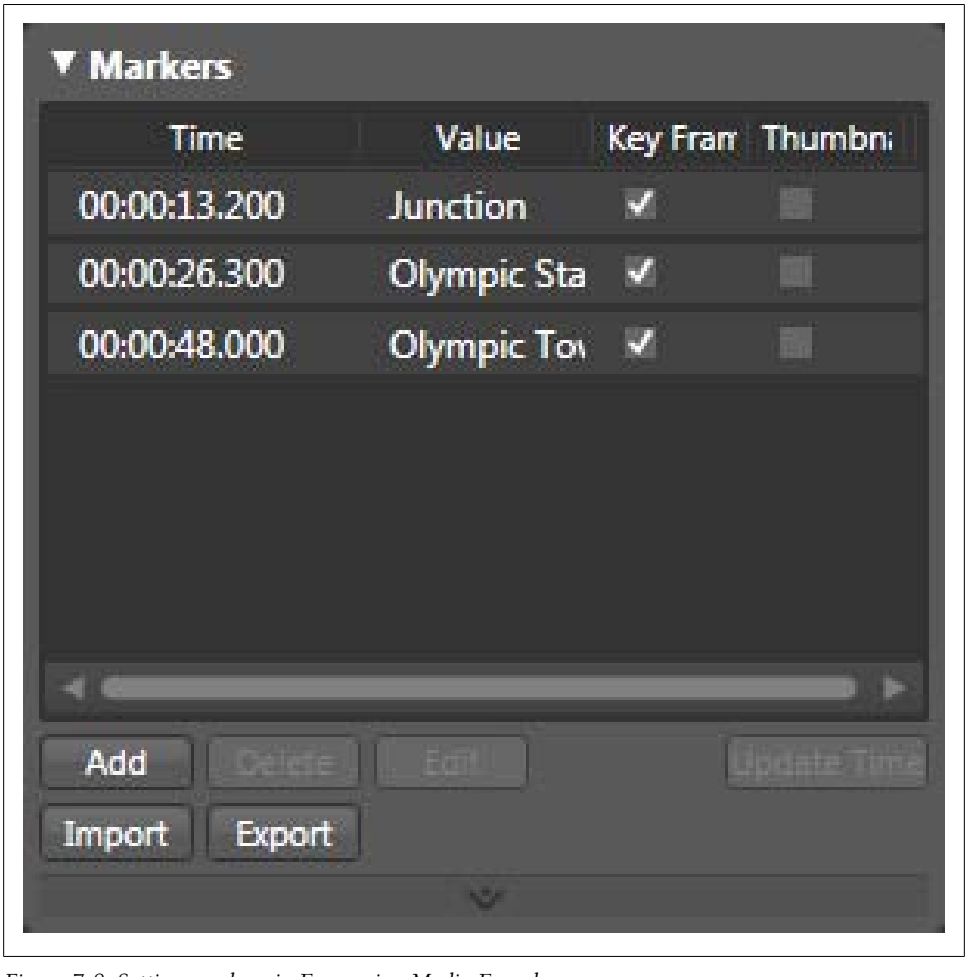


Figure 7-8. Setting markers in Expression Media Encoder

We will now successively add features to the player. The file names remain the same, but the number of features grows. In the downloadable code for this book you get the final version (see *http://www.oreilly.com/catalog/9780596516116*).

The Silverlight SDK contains a complete list of `<MediaElement>` attributes. A number of them will be introduced throughout the remainder of this chapter, but some of them are of general interest and will be discussed here:

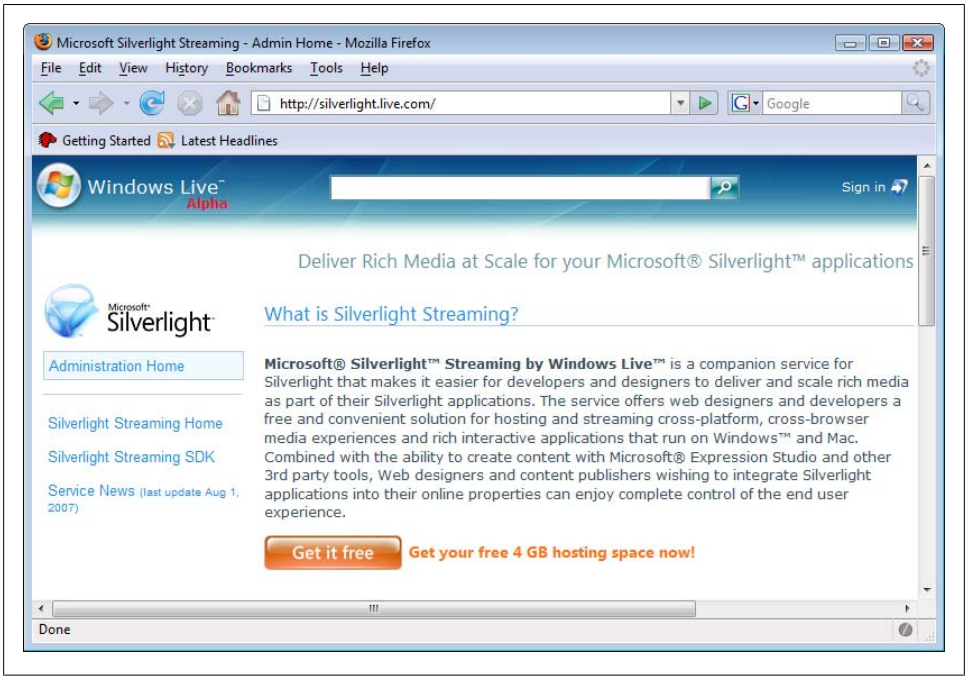


Figure 7-9. Streaming for Silverlight

`AutoPlay`
: By default, referenced media files are played as soon as the application has been fully loaded. You can disable this behavior by setting the attribute/property to `False`.

`Balance`
: Provides the audio volume ratio between left and right speaker. `-1` means that all audio is played in the left speaker only, `1` means that all audio is played in the right speaker only. The default value of `0` means that the audio is evenly shared between the left and the right speaker. You can choose any float values in between.

`IsMuted`
: Boolean value that depends on whether the audio output by the `<MediaElement>` is muted (`True`) or not (`False`).

`NaturalDuration`
: Duration of the media file; can only be read, not set

`NaturalVideoHeight`
: Height of the video, 0 for audio files; can only be read, not set

`NaturalVideoWidth`
: Width of the video, 0 for audio files; can only be read, not set

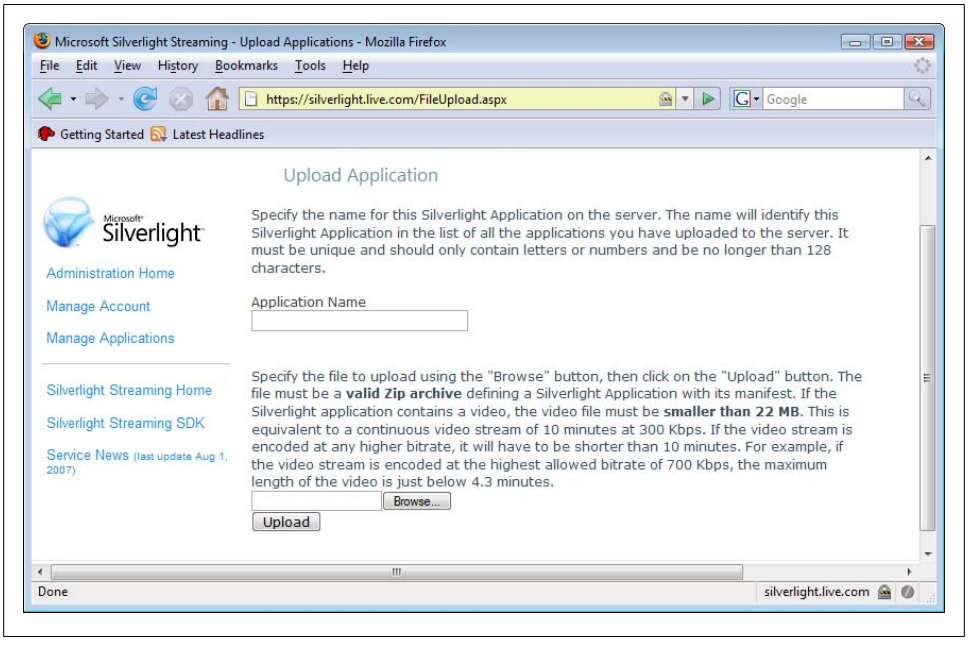


Figure 7-10. You may upload both full-blown applications and individual media content

`Stretch`
: Stretches the media if the display area is greater than the video size; the following options are available:

  `None`
  : Video size remains the original one

  `Fill`
  : The video fills up the whole available area, loosing its aspect ratio

  `Uniform`
  : The video size is increased, maintaining the aspect ratio, until either the video has the width or the height of the display area

  `UniformToFill`
  : The video size is increased, maintaining the aspect ratio, until the video width and height are both greater or equal than the width and height of the display area. If necessary, parts of the video are cropped.

`Volume`
: The audio volume as a value between `0` (muted) and `1` (maximum volume made available by the operating system). The default value is `0.5`.

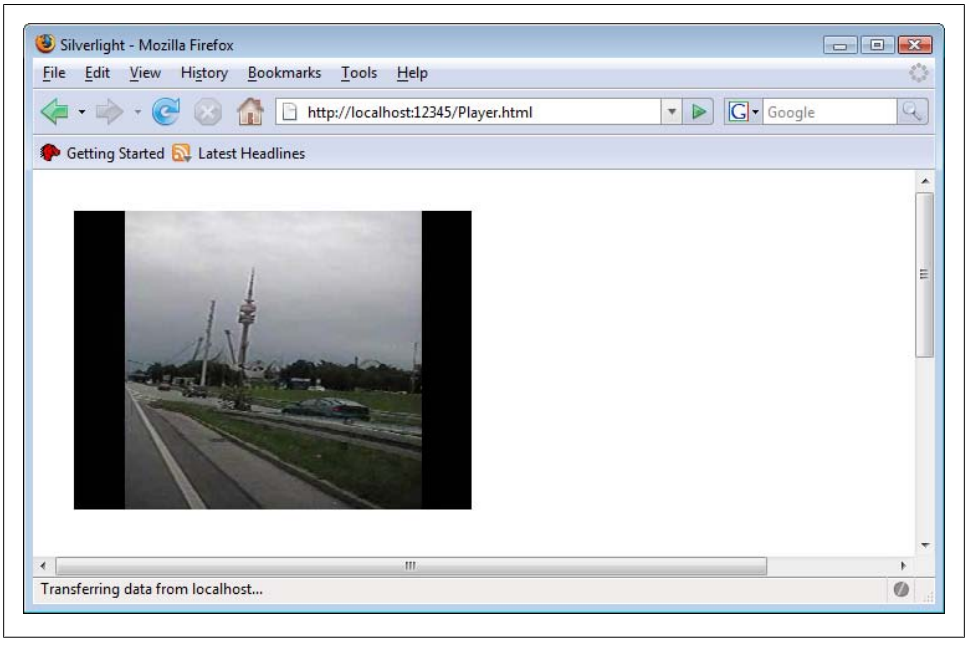
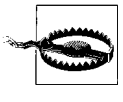


Figure 7-11. The (spartan) video player



When (programmatically) assigning a value to `Volume`, make sure that you always provide a string value. JavaScript can usually automatically convert numbers into strings, but the Silverlight API will complain.

Controlling Multimedia

`<MediaElement>` plays a media file. If it is audio, the sound is played. If it is video, the video is shown (and if there is sound within the video, the sound is played too). Apart from that, `<MediaElement>` does not have any kind of output or UI. Instead you need to create your own UI. Figure 7-12 shows a simple UI for our Silverlight media player. It was initially created in Expression Design, exported as XAML, imported into Expression Blend 2, and then tweaked and tuned. To keep the code size maintainable and printable, very basic structures were used to create the UI.

The UI consists of several buttons and other elements:

- A play and pause button (depending on the state of the media content)
- A button to jump to the previous marker
- A button to jump to the next marker
- Buttons to increase and decrease the volume
- A text containing the current volume level

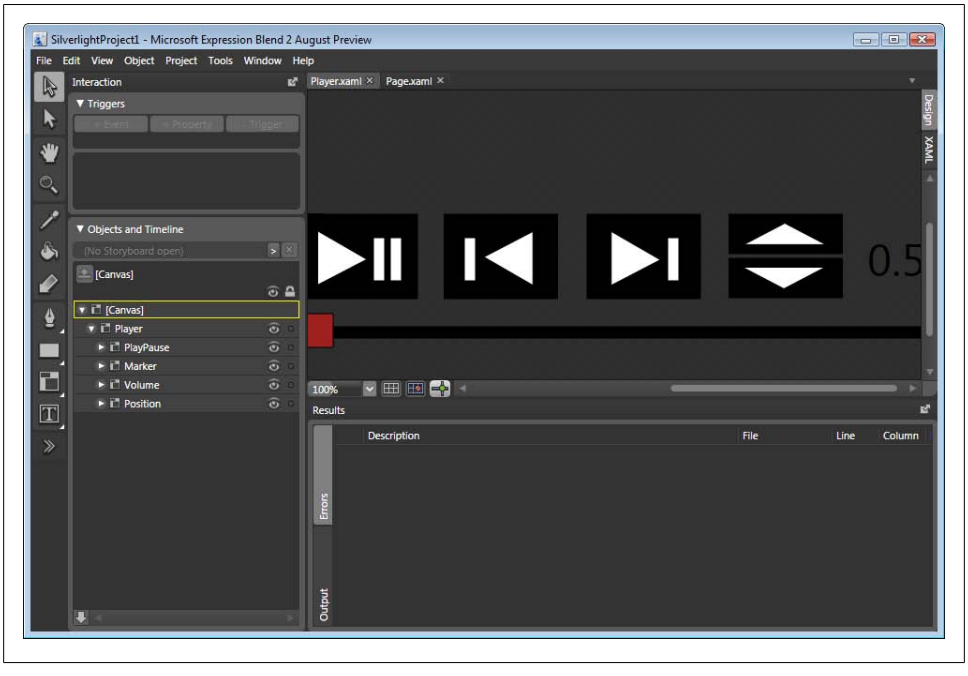


Figure 7-12. The SIlverlight player UI (loaded in Expression Blend 2)

- A timeline that serves as a progress bar for the media playback, including a pointer
- A text field which will later show the name of any marker (not visible at the moment)

The XAML file is structured so that every button (including `<Path>` elements on the button) is grouped in a `<Canvas>` element. We use attributes to assign event handlers. Example 7-2 shows the complete XAML markup for the UI. Don't worry whether the meaning of all the event handlers is obvious at the moment, here is a list of all of the assigned handler functions:

`playOrPause()`
: Called when the play/pause button is pressed; needs to play or pause the media content

`gotoPrevioiusMarker()`
: Called when the previous marker button is pressed; needs to seek to the position of the previous marker, if available

`gotoNextMarker()`
: Called when the next marker button is pressed; needs to seek to the position of the next marker, if available

`volumeUp()`
: Called when the upper volume button is pressed; needs to increase the volume by `0.1`, if possible

`volumeDown()`
: Called when the lower volume button is pressed; needs to decrease the volume by `0.1`, if possible

`showMarker()`
: Called when a marker is reached; needs to display the name of that marker

`initVideo()`
: Called when the video header has been loaded; needs to initialize the application and also start updating the progress bar pointer's position

More details on these event handlers will be covered when we write the associated code.

Example 7-2. The Silverlight media player, the XAML file (Player.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Canvas.Left="25" Canvas.Top="25">
  <Canvas x:Name="PlayerControls">
    <Canvas x:Name="PlayPause" MouseLeftButtonUp="playOrPause”>
      <Rectangle Width="122.014" Height="91.0105"
                 Canvas.Left="-0.50001" Canvas.Top="-0.5"
                 Stretch="Fill" StrokeLineJoin="Round"
                 Stroke="#FF000000" Fill="#FF000000"/>
      <Path x:Name="Path" Width="53.5274" Height="58.6534"
            Canvas.Left="13.9888" Canvas.Top="17.1706"
            Stretch="Fill" StrokeLineJoin="Round"
            Stroke="#FF000000" Fill="#FFFFFFFF"
            Data="F1 M 67.0161,47.9973L 14.4888,17.6706L 14.4888,75.324L
67.0161,47.9973 Z"/>
    <Rectangle Width="12.9994" Height="46.9976"
               Canvas.Left="73.5164" Canvas.Top="24.5136"
               Stretch="Fill" StrokeLineJoin="Round"
               Stroke="#FF000000" Fill="#FFFFFFFF"/>
    <Rectangle Width="12.9994" Height="46.9976"
               Canvas.Left="91.5155" Canvas.Top="24.5138"
               Stretch="Fill" StrokeLineJoin="Round"
               Stroke="#FF000000" Fill="#FFFFFFFF"/>
    </Canvas>
    <Canvas x:Name="Marker">
      <Canvas x:Name="PrevMarker" MouseLeftButtonDown="gotoPreviousMarker”>
        <Rectangle Width="122.014" Height="91.0105"
                   Canvas.Left="149.003" Canvas.Top="-0.5"
                   Stretch="Fill" StrokeLineJoin="Round"
                   Stroke="#FF000000" Fill="#FF000000"/>
        <Rectangle Width="12.9994" Height="46.9976"
                   Canvas.Left="170.511" Canvas.Top="24.5139"
                   Stretch="Fill" StrokeLineJoin="Round"
                   Stroke="#FF000000" Fill="#FFFFFFFF"/>
        <Path Width="53.5274" Height="58.6534"
              Canvas.Left="191.245" Canvas.Top="17.6856"
```

```
              Stretch="Fill" StrokeLineJoin="Round"
              Stroke="#FF000000" Fill="#FFFFFFFF"
              Data="F1 M 191.745,45.5123L 244.273,75.839L 244.273,18.1856L
191.745,45.5123 Z "/>
      </Canvas>
      <Canvas x:Name="NextMarker" MouseLeftButtonDown="gotoNextMarker”>
        <Rectangle Width="122.014" Height="91.0105"
                   Canvas.Left="301.486" Canvas.Top="-0.5"
                   Stretch="Fill" StrokeLineJoin="Round"
                   Stroke="#FF000000" Fill="#FF000000"/>
        <Path Width="53.5273" Height="58.6534"
              Canvas.Left="327.24" Canvas.Top="17.6856"
              Stretch="Fill" StrokeLineJoin="Round"
              Stroke="#FF000000" Fill="#FFFFFFFF"
              Data="F1 M 380.267,48.5123L 327.74,18.1856L 327.74,75.8391L
380.267,48.5123 Z "/>
        <Rectangle Width="12.9994" Height="46.9976"
                   Canvas.Left="387.502" Canvas.Top="24.5138"
                   Stretch="Fill" StrokeLineJoin="Round"
                   Stroke="#FF000000" Fill="#FFFFFFFF"/>
      </Canvas>
      <TextBlock x:Name="MarkerName" FontFamily="Segoe UI" FontSize="24"
                 Foreground="#FF000000" Canvas.Left="50" Canvas.Top="150" />
    </Canvas>
    <Canvas x:Name="Volume">
      <Canvas x:Name="Up" MouseLeftButtonDown="volumeUp”>
        <Rectangle Width="122.014" Height="43.0129"
                   Canvas.Left="452.987" Canvas.Top="-0.5"
                   Stretch="Fill" StrokeLineJoin="Round"
                   Stroke="#FF000000" Fill="#FF000000"/>
        <Path Width="88.9956" Height="24.0292"
              Canvas.Left="468.496" Canvas.Top="8.87177"
              Stretch="Fill" StrokeLineJoin="Round"
              Stroke="#FF000000" Fill="#FFFFFFFF"
              Data="F1 M 512.994,9.37177L 468.996,32.401L 556.992,32.401L
512.994,9.37177 Z "/>
      </Canvas>
      <Canvas x:Name="Down" MouseLeftButtonDown="volumeDown”>
        <Rectangle Width="122.014" Height="43.0128"
                   Canvas.Left="452.987" Canvas.Top="46.5052"
                   Stretch="Fill" StrokeLineJoin="Round"
                   Stroke="#FF000000" Fill="#FF000000"/>
        <Path Width="88.9955" Height="24.0292"
              Canvas.Left="467.497" Canvas.Top="55.9971"
              Stretch="Fill" StrokeLineJoin="Round"
              Stroke="#FF000000" Fill="#FFFFFFFF"
              Data="F1 M 511.995,79.5264L 555.992,56.4971L 467.997,56.4971L
511.995,79.5264 Z "/>
      </Canvas>
      <TextBlock x:Name="VolumeText" FontFamily="Segoe UI" FontSize="48"
                 Text="0.5" Foreground="#FF000000"
                 Canvas.Left="590" Canvas.Top="10" />
    </Canvas>
    <Canvas x:Name="Position">
      <Rectangle x:Name="Timeline" Width="669.987" Height="12.9994"
```

```
                Canvas.Left="-0.5" Canvas.Top="119.509"
                Stretch="Fill" StrokeLineJoin="Round"
                Stroke="#FF000000" Fill="#FF000000"/>
      <Rectangle x:Name="Pointer" Width="32.0184" Height="36.9982"
                Canvas.Left="-0.5" Canvas.Top="105.51"
                Stretch="Fill" StrokeLineJoin="Round"
                Stroke="#FF000000" Fill="#FFA01F1F"/>
    </Canvas>
  </Canvas>
  <Canvas x:Name="Player" Canvas.Top="200">
    <MediaElement x:Name="Video" Source="video.wmv" AutoPlay="False"
                  MarkerReached="showMarker" MediaOpened="initVideo"/>
  </Canvas>
</Canvas>
```

Play and Pause

The application needs permanent access to the embedded video file. The best way to achieve this is to save a reference in a global variable whenever the first event handler fires. This will be the handler for the `MediaOpened` event (if a user clicks a button before the video has been loaded, the user's request cannot be fulfilled at that time). In the handler function, the global `video` variable will be filled:

```
var video = null;

function initVideo(sender, eventArgs) {
  if (video == null) {
    video = sender;
  }
  // ...
}
```

Silverlight supports several methods for controlling whether a given media content is played or not:

`pause()`
: Pauses the media content

`play()`
: Plays the media content if it is stopped, or resumes it if it is paused

`stop()`
: Stops the media content

Conveniently, you will not get any JavaScript errors if you try an invalid operation like playing media content that is already playing, or pausing stopped media content. So you do not need any error handling in that respect.

The current state of the movie can be determined by accessing its `currentState` property. The following values are possible:

`Buffering`
: Video is loading, but not enough data has been streamed yet, so the control is buffering the data

`Closed`
: Video has been closed

`Error`
: There has been an error loading (or playing) the video

`Opening`
: The video is currently being opened

`Paused`
: The video is paused

`Playing`
: The video is being played

`Stopped`
: The video has been stopped

If the video is paused or stopped, JavaScript tries to play it; otherwise, JavaScript tries to pause it:

```
function playOrPause(sender, eventArgs) {
  if (video.currentState == 'Paused' ||
      video.currentState == 'Stopped') {
    video.play();
  } else {
    video.pause();
  }
}
```

And indeed, if you include this code in your "XAML code-behind" JavaScript file (and also provide empty shells for the other event handler functions), a click on the movie will play it (note that it does not start automatically, thanks to `AutoPlay="False"`), a second click will pause it, and a third click will resume it.

Setting the Volume

The volume of a media file is available using the aforementioned `volume` property (remember that JavaScript properties are identical to XAML attributes, but lower camelcase by convention). So when a user clicks on the upper volume button, increase the volume by 0.1; clicking on the lower button decreases the volume by 0.1. In theory, that's easy, but you do have to take some extra precautions. First of all, you need to make sure that the new volume is within the valid interval (from 0 through 1). Silverlight is not very forgiving when you supply invalid values here. Therefore, make sure first that the value is correct. For instance, when increasing the volume, determine the old volume, add 0.1, and then check whether this is greater than 1.0 or not, like this:

```
var newVolume = Math.min(1.0, video.volume + 0.1);
```

Then you have values like 0.5, 0.6, 0.7, or not. JavaScript is not very exact when it comes to floating point values, so you will rather get values like 0.5, 0.60000002, 0.69999986, and so on. To avoid any ugly values being used (and displayed), we round it to the first decimal value:

```
newVolume = Math.round(10 * newVolume) / 10;
```

Now we can set the new volume:

```
video.volume = newVolume.toString()
```

Finally, we display the new volume in the text field. To get rid of any extraneous decimal digits, we cut off everything after the first decimal digit:

```
sender.findName('VolumeText').text = newVolume.toString().substring(0, 3);
```

Here is the complete code for both the `volumeUp()` and the `volumeDown()` functions:

```
function volumeUp(sender, eventArgs) {
  var newVolume = Math.min(1.0, video.volume + 0.1);
  newVolume = Math.round(10 * newVolume) / 10;
  video.volume = newVolume.toString()
  sender.findName('VolumeText').text = newVolume.toString().substring(0, 3);
}

function volumeDown(sender, eventArgs) {
  var newVolume = Math.max(0.0, video.volume - 0.1);
  newVolume = Math.round(10 * newVolume) / 10;
  video.volume = newVolume.toString()
  sender.findName('VolumeText').text = newVolume.toString().substring(0, 3);
}
```

When you run the code in the browser, you will actually hear that your code works when clicking on the volume buttons, and you will also see the new volume values, as Figure 7-13 shows.

Determining the Media Position

So, playing and pausing a video is relatively trivial, but determining its current position requires an extra step. There is indeed a property that exposes this information: `position`. However, this property is of type `TimeSpan`, and the string representation is something like 12:34:56 (12 hours, 34 minutes, 56 seconds). This data has two disadvantages:

- It is hard to calculate, e.g., What percentage of the movie has already been played?
- You cannot create `TimeSpan` values with JavaScript, so you cannot directly set the `position` property.

There is a good workaround for both issues. A sub-property of any `TimeSpan` value is seconds, which converts the `TimeSpan` value into a float value, the number of seconds (not necessarily integral). You can read out this value and set it, which is covered in "Working with Markers.

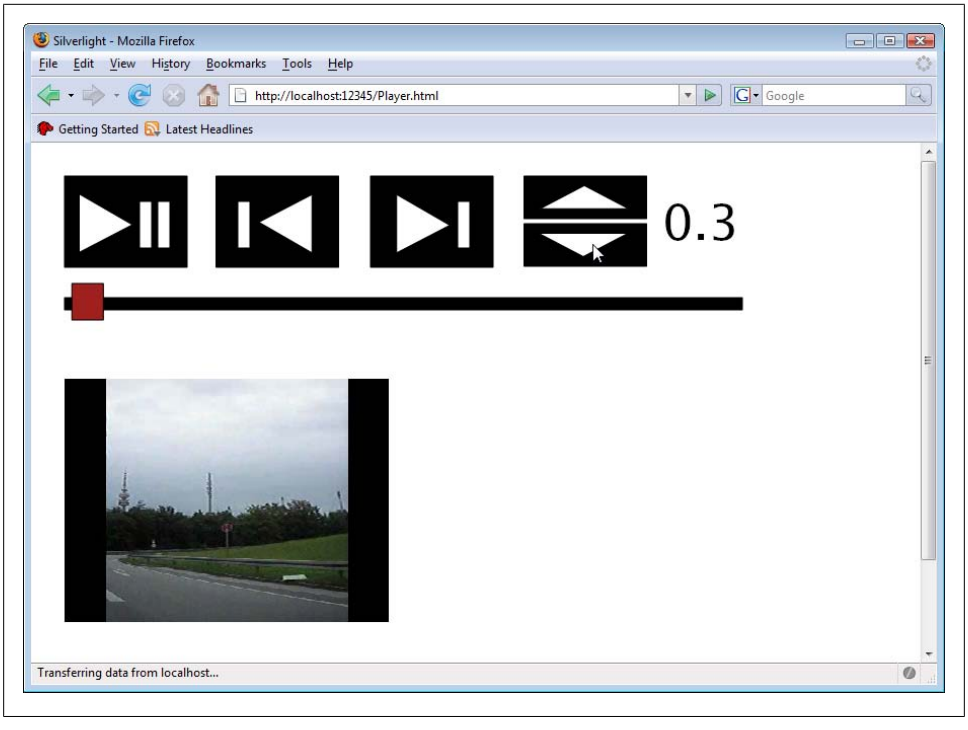


Figure 7-13. The Silverlight media player can now control the volume

But before that, we will tackle the progress bar. We need to look up the current position of the media with is`video.position.seconds`. Then we need to determine the length of the media with `video.currentPosition.seconds`. The `currentPosition` property also returns a `TimeSpan` value, so accessing the `seconds` subproperty yields the desired result.

By dividing the current position by the length of the media, we have a percentage of how much has already been played, say, 30 percent. Consequently, we need to put the progress bar pointer at 30 percent of the progress bar length! Or, to be exact, the center of the progress bar pointer needs to be at 30 percent. If the pointer has a width of 30 pixels, we need to place it at 30 percent of the progress bar and then move it 15 pixels (half the length) to the left. Generally speaking, this code calculates the correct position of the pointer, relative to the progress bar:

```
var currentPosition = video.position.seconds;
var length = video.naturalDuration.seconds;
var progressBar = video.findName('Timeline');
var progressPointer = video.findName('Pointer');
var relativePosition = (currentPosition/length) * progressBar.width -
                       (progressPointer.width / 2);
```

The relative position needs to be converted into an absolute position within the surrounding `<Canvas>` element. The easiest way to achieve this is to determine the position of the progress bar and then just add the `relativePosition` value:

```
progressPointer['Canvas.Left'] = progressBar['Canvas.Left'] + relativePosition;
```

We need to make sure that the position is updated again (and again and again). Therefore, we create a JavaScript timeout that calls the function again in one second (a thousand milliseconds). Here is the complete code for this function:

```
function updatePosition(sender, eventArgs) {
  var currentPosition = video.position.seconds;
  var length = video.naturalDuration.seconds;
  var progressBar = video.findName('Timeline');
  var progressPointer = video.findName('Pointer');
  var relativePosition = (currentPosition/length) * progressBar.width -
                         (progressPointer.width / 2);
  progressPointer['Canvas.Left'] = progressBar['Canvas.Left'] + relativePosition;
  setTimeout(updatePosition, 1000);
}
```

In the `initVideo()` function, `updatePosition()` is then called for the first time.

You may want to shorten the interval in which the pointer position is updated, so that the pointer moves smoothly over the progress bar.

Figure 7-14 shows the result. The progress pointer is moving as the movie is playing along.

Working with Markers

A marker defines a certain spot within a media file. There are at least two UI approaches for markers. One is to notify the user when a marker has been reached, and the other is to offer to let the user to jump between markers. Let's start with the first one. The `MediaElement` element supports the `MarkerReached` event. When, during playing a media file, a marker is reached this event is fired. As always, the event handling function passes two arguments, the sending object (the `MediaElement`) and event information. The latter argument is of great interest here: its marker property provides three fields from the marker:

`text`
: The name of the marker

`time`
: The time of the marker (again as a `TimeSpan` value)

`type`
: The type of the marker (depending on which marker types are supported by the format used)

The XAML file already contains a text field (named `MarkerName`) for the name of the marker. Whenever a marker is reached, its name is written into that field:

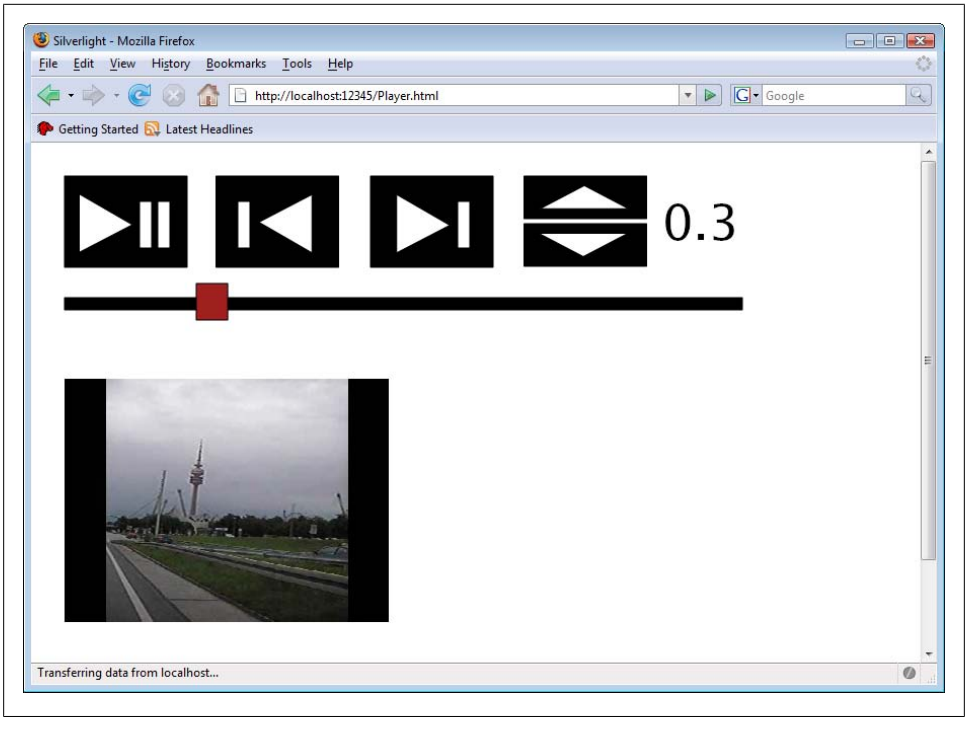


Figure 7-14. The movie's progress is displayed

```
function showMarker(sender, eventArgs) {
  sender.findName('MarkerName').text = eventArgs.marker.text;
  ...
}
```

However, it would be a bad idea usability-wise to let this information stay where it is. A few seconds later, the name of the marker is maybe not suitable any more. Therefore, we remove this text after two seconds by using a JavaScript timeout:

```
markerTimeout = setTimeout(
  function() {
    video.findName('MarkerName').text = '';
    markerTimeout = null;
  }, 2000);
```

What happens if there are two markers within less than two seconds? Imagine that there is one marker at 00:00:02, and one at 00:00:03. The appearance of the first marker sets a timeout that will be fired at 00:00:04; the second marker sets a timeout that will be fired at 00:00:05 (after two seconds each). However, the 00:00:04 timeout will empty the text field, which has just been populated by the second marker at 00:00:03! Therefore, we need to check first whether there is a pending timeout. If so, we have to delete it (the timeout, not the text field), using the `clearTimeout()` JavaScript method:

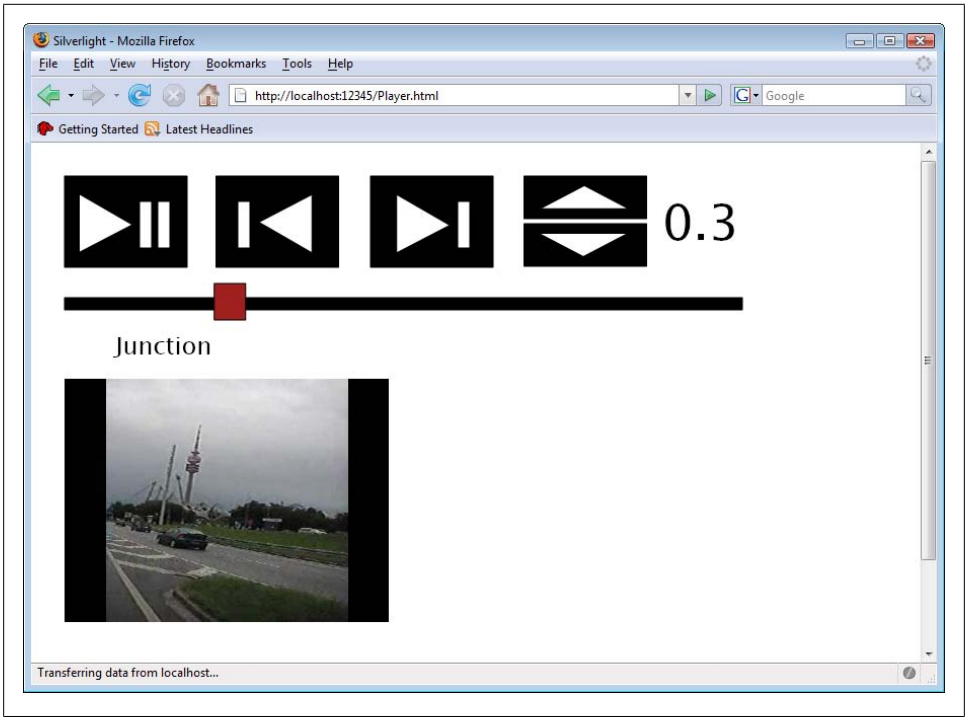


Figure 7-15. A marker has been reached

```
if (markerTimeout != null) {
  clearTimeout(markerTimeout);
}
```

Here is the complete code for the `showMarker()` function:

```
function showMarker(sender, eventArgs) {
  sender.findName('MarkerName').text = eventArgs.marker.text;
  if (markerTimeout != null) {
    clearTimeout(markerTimeout);
  }
  markerTimeout = setTimeout(
    function() {
      video.findName('MarkerName').text = '';
      markerTimeout = null;
    }, 2000);
}
```

As Figure 7-15 shows, the marker names are displayed once a marked position in the movie has been reached.

The final item on our to do list is also related to markers: remember that we still have functionless buttons that allow users to jump to the next or previous marker? To implement this functionality, we first have to determine all the markers. We could also look up all markers once one of the buttons is pressed, but then we would repeat our

efforts. Therefore, the markers are retrieved once the movie has started loading. Once the `MediaOpened` event has been fired, we have access to all markers. The embedded media's markers property provides us with a collection of all markers. Such a collection has a number of value properties, and we need the following ones:

`count`
: The number of items in the collection

`getItem()`
: Returns the collection item with the given position

Every collection item is a marker with the aforementioned properties `text`, `time`, and `type`. We are just interested in the marker time and save this information in a global JavaScript array. At the end, we numerically sort the array, just to make sure that the marker times are in the correct position:

```
var markerTimes = [];

function loadMarkers(sender, eventArgs) {
  for (var i = 0; i < video.markers.count; i++) {
    markerTimes[markerTimes.length] = video.markers.getItem(i).time;
  }
  markerTimes.sort(function(a, b) { return a.seconds - b.seconds; });
}
```

The `loadMarkers()` function is called in the `MediaOpened` event handler, the `initVideo()` function.

First we tackle the `goNextMarker()` function. We determine the current position of the clip and then go through all the markers. If we find a marker that is later than the current media position (assuming that all markers are sorted by their time), we use this marker and immediately leave the function. Here is the code:

```
function gotoNextMarker(sender, eventArgs) {
  var currentTime = video.position.seconds;
  for (var i = 0; i < markerTimes.length; i++) {
    if (markerTimes[i].seconds > currentTime) {
      video.position = markerTimes[i];
      break;
    }
  }
}
```

The function `gotoPreviousMarker()` is implemented in analogous fashion. Make sure you start searching from the end. We go through the markers starting with the last one, until we find one that is earlier than the current media position:

```
function gotoPreviousMarker(sender, eventArgs) {
  var currentTime = video.position.seconds;
  for (var i = markerTimes.length - 1; i >= 0; i--) {
    if (markerTimes[i].seconds < currentTime) {
      video.position = markerTimes[i];
      break;
    }
```

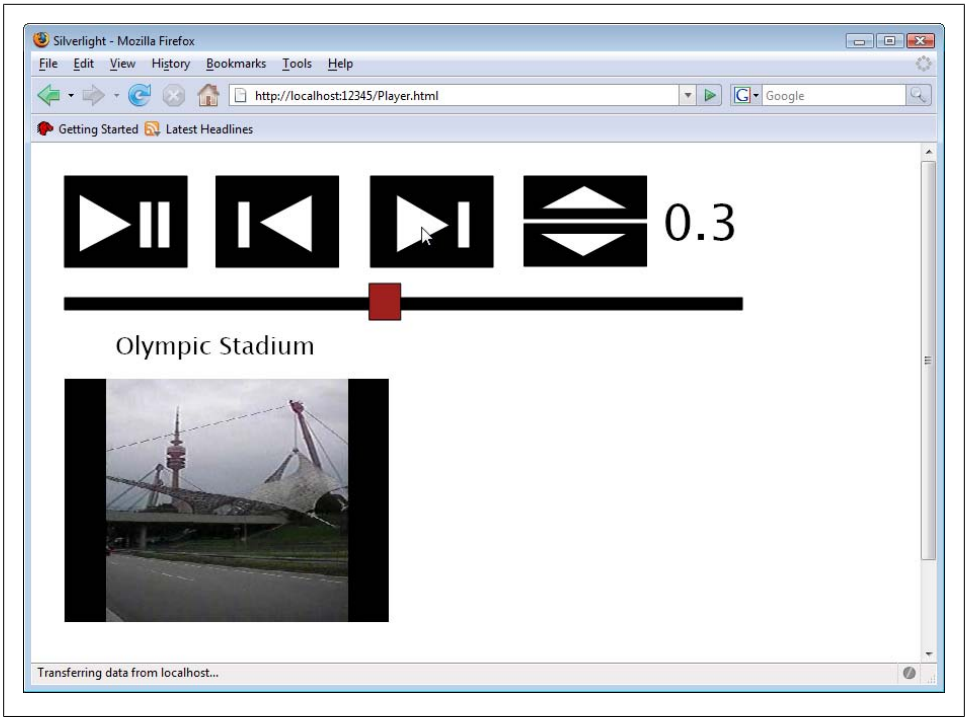
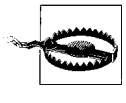


Figure 7-16. Clicking on the button jumps to the next marker

```
    }
}
```

If you run this example in the browser, you will notice that the player jumps to the appropriate position in the clip (see Figure 7-16), but only if the media format supports jumping to markers. If not, you conveniently do not get an error message.



In the current Silverlight version, a (possible) bug prevents setting `video.position.seconds` (setting the value has no effect). In our special example, we can avoid this issue since we save the `TimeSpan` marker values in an array and can then assign the `TimeSpan` values back to `video.position` when jumping to a marker.

Every feature implemented in this chapter has not been extraordinarily difficult to understand and if you combine them, you will get a very functional application. Example 7-3 contains the complete JavaScript code for the Silverlight media player.

Example 7-3. The SIlverlight media player, the XAML JavaScript file (Player.xaml.js)

```
var video = null;
var markerTimes = [];
var markerTimeout = null;
```

```
function playOrPause(sender, eventArgs) {
  if (video.currentState == 'Paused' ||
      video.currentState == 'Stopped') {
    video.play();
  } else {
    video.pause();
  }
}

function gotoPreviousMarker(sender, eventArgs) {
  var currentTime = video.position.seconds;
  for (var i = markerTimes.length - 1; i >= 0; i--) {
    if (markerTimes[i].seconds < currentTime) {
      video.position = markerTimes[i];
      break;
    }
  }
}

function gotoNextMarker(sender, eventArgs) {
  var currentTime = video.position.seconds;
  for (var i = 0; i < markerTimes.length; i++) {
    if (markerTimes[i].seconds > currentTime) {
      video.position = markerTimes[i];
      break;
    }
  }
}

function volumeUp(sender, eventArgs) {
  var newVolume = Math.min(1.0, video.volume + 0.1);
  newVolume = Math.round(10 * newVolume) / 10;
  video.volume = newVolume.toString()
  sender.findName('VolumeText').text = newVolume.toString().substring(0, 3);
}

function volumeDown(sender, eventArgs) {
  var newVolume = Math.max(0.0, video.volume - 0.1);
  newVolume = Math.round(10 * newVolume) / 10;
  video.volume = newVolume.toString()
  sender.findName('VolumeText').text = newVolume.toString().substring(0, 3);
}

function showMarker(sender, eventArgs) {
  sender.findName('MarkerName').text = eventArgs.marker.text;
  if (markerTimeout != null) {
    clearTimeout(markerTimeout);
  }
  markerTimeout = setTimeout(
    function() {
      video.findName('MarkerName').text = '';
      markerTimeout = null;
    }, 2000);
}
```

```
function initVideo(sender, eventArgs) {
  if (video == null) {
    video = sender;
  }
  loadMarkers(sender, eventArgs);
  updatePosition(sender, eventArgs);
}

function loadMarkers(sender, eventArgs) {
  for (var i = 0; i < video.markers.count; i++) {
    markerTimes[markerTimes.length] = video.markers.getItem(i).time;
  }
  markerTimes.sort(function(a, b) { return a.seconds - b.seconds; });
}

function updatePosition(sender, eventArgs) {
  var currentPosition = video.position.seconds;
  var length = video.naturalDuration.seconds;
  var progressBar = video.findName('Timeline');
  var progressPointer = video.findName('Pointer');
  var relativePosition = (currentPosition/length) * progressBar.width -
                         (progressPointer.width / 2);
  progressPointer['Canvas.Left'] = progressBar['Canvas.Left'] + relativePosition;
  setTimeout(updatePosition, 1000);
}
```

Storing Markers

Usually markers are stored within a media file. However, you can also temporarily create markers. Chapter 10 will show you how this can be done using the ASP.NET Futures. You can also use JavaScript by dynamically adding `<TimelineMarker>` elements to the `<MediaElement>` element (you cannot do this via regular markup). The trick is to dynamically create a XAML object with the `createFromXaml()` method, and then add this to the `markers` collection of the media content. This is how the code could look like:

```
function initVideo(sender, eventArgs) {
  var timeLineMarker = sender.getHost().content.createFromXaml(
    '<TimelineMarker Time="12:34:56" Text="Olympic Stadium" />');
    sender.markers.add(timeLineMarker);
}
```

Chapter 8 will provide you with more information on how to use JavaScript to access Silverlight content embedded in a page.

Using audio and video from Silverlight is quite convenient. Just convert your content into a supported file format and use `<MediaElement>` markup. With a little bit of JavaScript you can turn your UI into a full-fledged media player; but, obviously, if it came out of the box that way it would be even more convenient.

For Further Reading

http://www.microsoft.com/expression/products/overview.aspx?key=encoder
: Information on the Microsoft Expression Encoder

PART III

Programmatic Silverlight

CHAPTER 8

Accessing Silverlight Content From JavaScript

JavaScript, the Browser Language

The first parts of this book were called "Declarative Silverlight," but they contained a lot of JavaScript code, like all event handlers. Our focus shifted to add JavaScript to the XAML, which is why most JavaScript files are called *<something>.xaml.js*.

This chapter introduces a different approach. We will create a number of *<something>.html.js* files, code-behind JavaScript files for the HTML document containing the Silverlight content, so to speak. The JavaScript code in there will access the Silverlight content, add new Silverlight elements, and read information about the plug-in or the content of the XAML file.

Accessing the Plug-in

To access Silverlight content embedded on a page, you first need to access the plug-in. There are two ways to retrieve this information: access the plug-in from within the XAML event handler code or use the JavaScript DOM (Document Object Model). Let's start with the latter option and look at the, by now, well-known JavaScript code to load Silverlight content:

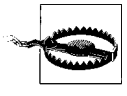
```
function createSilverlight()
{
  Silverlight.createObjectEx({
    source: 'Info.xaml',
    parentElement: document.getElementById('SilverlightPlugInHost'),
    id: 'SilverlightPlugIn',
    properties: {
      width: '400',
      height: '300',
      background:'#ffffffff',
      isWindowless: 'false',
      version: '1.0'
```

```
    },
    events: {
      onError: null,
    }
  });
}
```

Note the `id` property. This provides the DOM ID JavaScript can use to access the plugin:

```
var plugin = document.getElementById('SilverlightPlugIn');
```

If you are using ASP.NET AJAX, you can, of course, save some typing and use `$get()` instead of `document.getElementById()`.



It is tempting to use the ID of the `<div>` element holding the Silverlight content (in all of this book's examples: `SilverlightPlugInHost`), but this will not access the plug-in itself.

Then, starting with the `plugin` variable, you can access quite a number of data on the plug-in and its contents, but we will come back to that in a minute. First we will have a look at the other option to access the plug-in, from within the XAML event handling code.

Every object in the XAML has a (JavaScript) method called `getHost()` that also returns a reference to the plug-in. Assume that the `eventHandler()` function handles any event for the XAML file, then this code would appropriately fill the `plugin` variable:

```
function eventHandler(sender, eventArgs) {
  var plugin = sender.getHost();
}
```

One you have accessed the plug-in, you can go further. The Silverlight plug-in exposes three kinds of information to JavaScript:

General plug-in information
: These are accessible as direct properties or methods of the plug-in object. Examples are `source` (the XAML source code), `initParams` (the set of options used when initializing the Silverlight plug-in in the `createSilverlight()` function), and `onError` (the event handler that handles errors).

Plug-in settings information
: These are accessible using `plugin.settings.<property>`. Examples are `background` (the background color of the current Silverlight content) and `maxFrameRate` (the maximum frame rate in frames per second).

Plug-in content information
: These are accessible using `plugin.content.<property>`. Examples are `findName()` (the already known method to find XAML elements by their names), `fullScreen` (whether to display the content in full screen), and `root` (the `root` canvas element of the Silverlight content).

This chapter will showcase some of the most interesting options. For a complete list of the APIs, refer to Appendix A.

Communicating with the Plug-in

Once your JavaScript code has access to the plug-in, you can find details about the plug-in and its configuration and also relatively flexible access to the Silverlight content too. This section features some scenarios and examples.

Determining Plug-in Settings

The first example will determine some of the plug-in's settings. It will also feature both access methods by using the JavaScript DOM and the JavaScript code in the XAML code-behind. First, we need the containing HTML page, where we load the Silverlight content. A button on the page will be used to trigger the retrieval and display of plug-in information. Example 8-1 shows the code.

Example 8-1. Displaying plug-in information, the HTML file (Info.html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Silverlight</title>

  <script type="text/javascript" src="Silverlight.js"></script>
  <script type="text/javascript" src="Info.html.js"></script>
  <script type="text/javascript" src="Info.xaml.js"></script>
</head>

<body>
  <div id="SilverlightPlugInHost">
    <script type="text/javascript">
      createSilverlight();
    </script>
  </div>
  <div>
    <form action="">
      <input type="button" value="Show plugin info" onclick="showInfoJS();" />
    </form>
  </div>
</body>
</html>
```

In the "HTML code-behind" *Info.html.js*, the `showInfoJS()` function accesses the plug-in using the DOM, and then calls another function called `showInfo()`. This latter function will be implemented later on.

```
function showInfoJS() {
  var plugin = document.getElementById('SilverlightPlugIn');
```

```
  showInfo(plugin);
}
```

Example 8-2 shows the XAML file, without any complicated XAML markup, but there is an event handler for the left mouse button attached to the canvas.

Example 8-2. Displaying plug-in information, the XAML file (Info.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        MouseLeftButtonDown="showInfoXaml">
  <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15" />
  <TextBlock FontFamily="Arial" FontSize="56" Canvas.Left="25" Canvas.Top="40"
             Foreground="Black" Text="Silverlight" />
</Canvas>
```

The JavaScript file associated to the XAML file, displayed in Example 8-3, determines the plug-in and submits it as an argument to `showInfo()`, which is the same function that *Info.html.js* is using.

Although we are always referring to the JavaScript files as "XAML code-behind" and "HTML code-behind" to make a clear distinction, there is no difference between them from a browser's point of view. Both kinds of JavaScript files are loaded and executed; thus, code from one file may call code from another file.

Example 8-3. Displaying plug-in information, the XAML JavaScript file (Info.xaml.js)

```
function showInfoXaml(sender, eventArgs) {
  var plugin = sender.getHost();
  showInfo(plugin);
}
```

Finally, the `showInfo()` function needs to be implemented. It takes the plug-in reference, accesses two information points, and outputs them using the JavaScript `alert()` function. Example 8-4 has the code and when you click on either the Silverlight content or the HTML button, you will get an output similar to the one in Figure 8-1.

Example 8-4. Displaying plug-in information, the HTML JavaScript file (Info.html.js; excerpt)

```
function showInfoJS() {
  var plugin = document.getElementById('SilverlightPlugIn');
  showInfo(plugin);
}

function showInfo(plugin) {
  var s = 'Background: ' + plugin.settings.background;
  s += '\nMaxFrameRate: ' + plugin.settings.maxFrameRate;
  alert(s);
}
```

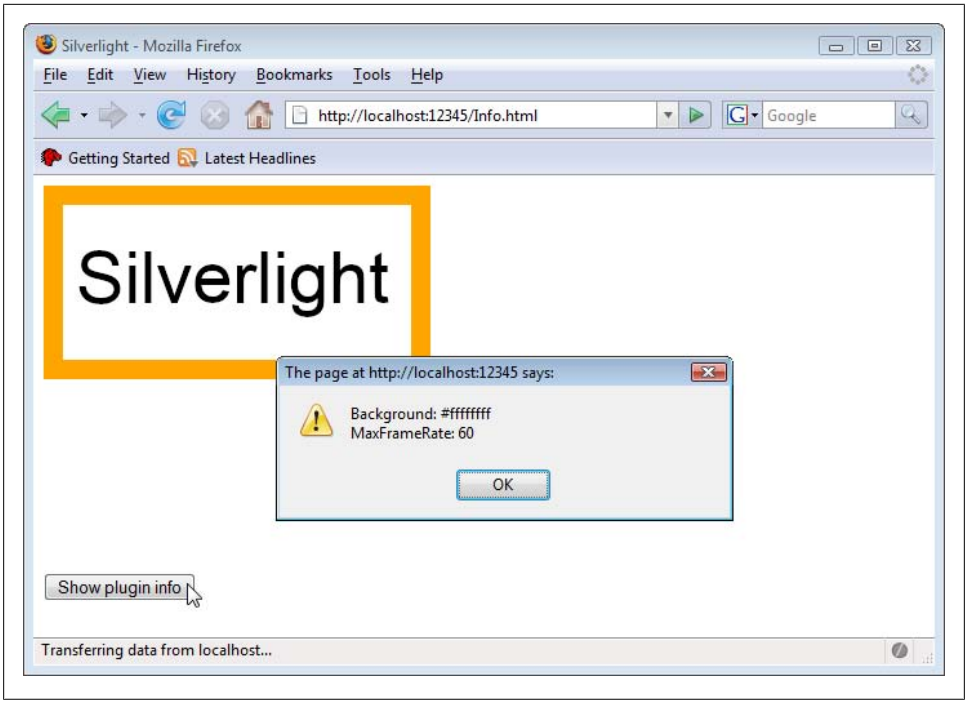


Figure 8-1. Displaying plug-in information

Modifying XAML Content

The plug-in's `source` property (`plugin.content.source`, to be exact) not only retrieves the XAML markup of the currently loaded Silverlight content, but also sets this information. This allows us to create a sample application with a similar concept as the WPF content viewer XAMLPad mentioned in Chapter 1. An HTML text box enables users to enter XAML markup; JavaScript code then tries to load this markup into the Silverlight plug-in. Let's start with the HTML and the input field, as shown in Example 8-5.

Example 8-5. SilverlightPad, the HTML file (SilverlightPad.html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Silverlight</title>

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

<body>
  <div>
    <div id="SilverlightPlugInHost">
      <script type="text/javascript">
```

```
        createSilverlight();
      </script>
    </div>
    <div id="InputHost">
      <textarea id="XamlMarkup" style="width:600px;" rows="10"></textarea><br />
      <input type="button" value="Update!" onclick="update();" />
    </div>
  </div>
</body>
</html>
```

The initial XAML markup is quite simple. In this example, it is the task of the user to provide some fancy Silverlight content after all. Example 8-6 has the code.

Example 8-6. SilverlightPad, the XAML markup (SilverlightPad.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Rectangle Width="600" Height="200" Stroke="Orange" StrokeThickness="15" />
  <Canvas Canvas.Left="30" Canvas.Top="30" x:Name="XamlOutput">
    <TextBlock FontFamily="Arial" FontSize="32" Foreground="Black"
               Text="Enter data below ..."/>
  </Canvas>
</Canvas>
```

The most sophisticated part of this example is certainly the JavaScript code. However, once you know how to do it, you just need a couple of lines. First, convert the XAML string provided by the user into a XAML JavaScript object that can work with the Silverlight plug-in. The `plug-in.content.createFromXaml()` method does exactly that:

```
var plugin = document.getElementById('SilverlightPlugIn');
var obj = plugin.content.createFromXaml(
  document.getElementById('XamlMarkup').value);
```

If the XAML is not valid, `obj` is `null`, so we can act accordingly. If on the other hand `obj` is not `null`, the user-supplied XAML is valid and can be applied to the page.

If you have a look back at Example 8-6 you will see that there is a second `<Canvas>` element within the root `<Canvas>`. The idea is that the outer (orange) border will always be there, whereas the content of that rectangle may be changed by the user. Therefore, the JavaScript code first needs to access the inner `<Canvas>`. This can be done by using `findName()`:

```
var XamlOutput = plugin.content.findName('XamlOutput');
```

`XamlOutput`—and any other XAML JavaScript object—does not support the HTML DOM, but a similar API accesses all child elements: `XamlOutput.children`. The following properties and methods exist:

`add()`
: Adds a child to the end of the list

`clear()`
: Empties the children list

`count`
: Provides the number of children

`getItem()`
: Retrieves the child with a given index

`getValue()`
: Retrieves the value of a given property

`insert()`
: Inserts a child (second argument) at a given index (first argument)

`name`
: Displays the name of the child (if available)

`remove()`
: Removes a given child from the list

`removeAt()`
: Removes a child with a given index

`setValue()`
: Sets the value of a given property

For this example, we only need to do two things: delete all children of the inner `<Canvas>` element (imagine that a user has already submitted XAML that has been rendered in the plug-in) and add the XAML JavaScript object (`obj`) to the children list:

```
 if (obj != null) {
  XamlOutput.children.clear();
  XamlOutput.children.add(obj);
} else {
  alert('Error! XAML might not be valid.');
}
```

Example 8-7 shows the complete JavaScript code (as always, minus the `createSilverlight()` function), and Figure 8-2 presents SilverlightPad in action. The XAML entered in the text field is displayed within the orange border.

Example 8-7. SilverlightPad, the HTML JavaScript file (SilverlightPad.html.js; excerpt)

```
function update() {
  var plugin = document.getElementById('SilverlightPlugIn');
  with (plugin.content) {
    var XamlOutput = findName('XamlOutput');
    var obj = createFromXaml(
      document.getElementById('XamlMarkup').value);
  }
  if (obj != null) {
    XamlOutput.children.clear();
    XamlOutput.children.add(obj);
  } else {
    alert('Error! XAML might not be valid.');
  }
}
```

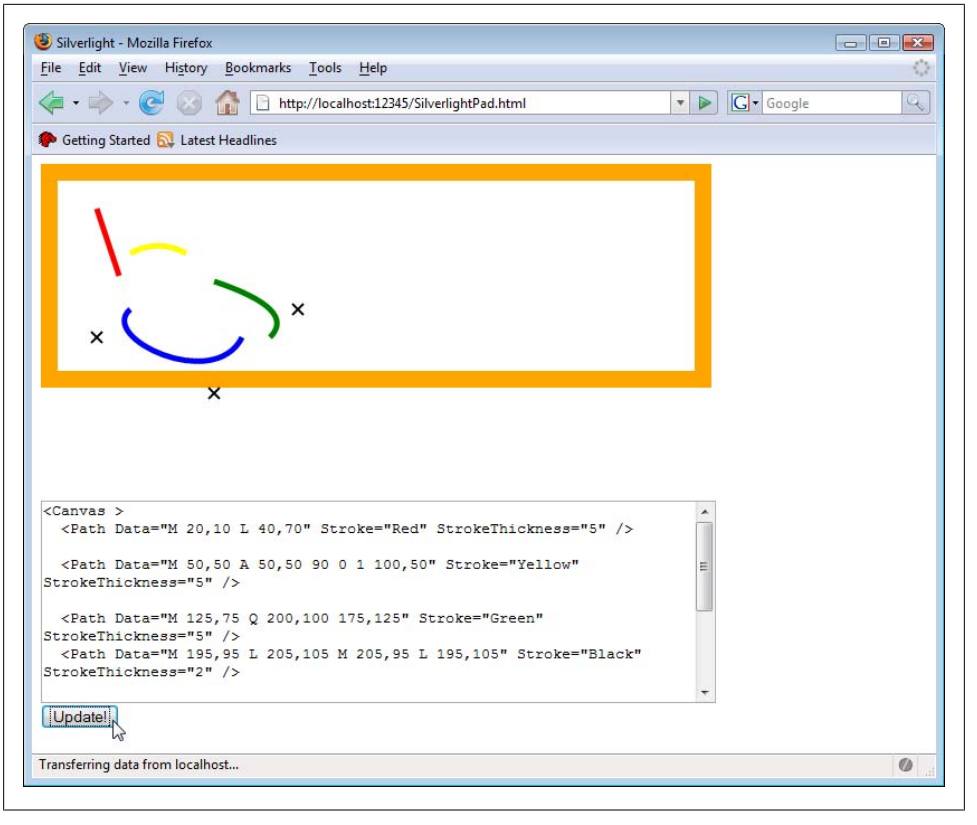


Figure 8-2. SilverlightPad

There is a simpler implementation. Instead of using the pseudo DOM of Silverlight, you can just set the `source` property of the plug-in.

Dumping Content Information

As a final sample application for this chapter, we will create a development helper tool that dumps information about a given Silverlight application. The sample features both dynamic loading of Silverlight content and also recursively accessing all elements. As usual, we start with the HTML file (see Example 8-8). But there is a surprise—there is no `createSilverlight()` call, only the empty `<div>` container `SilverlightPlugInHost`. Additionally, there are three more HTML UI widgets on the page:

- A text field (`XamlFile`) where users may enter the path to a XAML file
- A button that triggers loading the file

- A button that triggers dumping the file's contents. This button is initially disabled (why dump data when no data has been loaded yet?)

The page also contains a yet empty `<div>` element called `OutputDiv` where the "data dump" will be later displayed.

Example 8-8. Dumping Silverlight content, the HTML file (SilverlightDumper.html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Silverlight</title>

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

<body>
  <form action="">
  <div>
    <div id="SilverlightPlugInHost">
    </div>
    <div id="LoadXamlFile">
      <input type="text" name="XamlFile" />
      <input type="button" value="Load file" onclick="loadFile(this.form);" />
      &mdash;
      <input type="button" id="DumpButton" value="Dump XAML" onclick="dump();"
             disabled="disabled" />
    </div>
    <div id="OutputDiv">
    </div>
  </div>
  </form>
</body>
</html>
```

In the "JavaScript code-behind," the `loadFile()` function is responsible for loading the XAML file the user has previously selected. First of all, the file name needs to be determined:

```
function loadFile(f) {
  var filename = f.elements['XamlFile'].value);
  ...
}
```

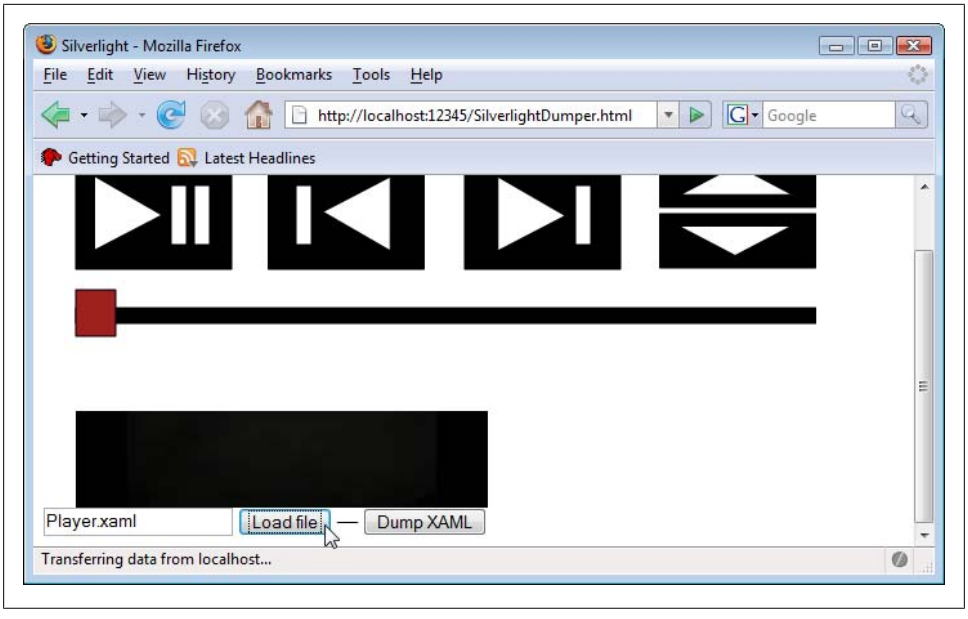


Figure 8-3. Dynamically loading a XAML file

Note that in Example 8-8, the text field does not have an `id` attribute, but just a `name` attribute. Therefore, you cannot use `document.getElementById()` to access the form element. However, you can use the JavaScript `document.forms[0].elements` array to read out the name. In this special case, we saved some typing. When we called `loadFile(this.form)` from the HTML page, a reference to the current form was submitted to `loadFile()`. Therefore, `f.elements['XamlFile']` grants access to the form field, and the `value` property contains the field's contents.

This file name is then provided to the well-known `createSilverlight()` function. Actually, it is provided to a modified `createSilverlight()` function. The modified version accepts the XAML file name to load as a parameter, allowing the application to dynamically load XAML files. The `loadFile()` function also enables the button that takes care of the (not yet implemented) dumping of Silverlight contents.

```
function loadFile(f) {
  createSilverlight(f.elements['XamlFile'].value);
  document.getElementById('DumpButton').disabled = false;
}
```

Figure 8-3 shows that the code so far already works. Enter a (relative) file path to an XAML file, and click the first button. The XAML content is displayed above the form.

The next step in our application is also the final one: dumping the contents of the currently loaded XAML file. In order to implement this, the code first needs to access the plug-in, as always:

```
var plugin = document.getElementById('SilverlightPlugIn');
```

We want to dump all elements, so we have to start at the top. The `root` property provides us with a reference to the root (`<Canvas>`) node:

```
var rootNode = plugin.content.root;
```

All that's left to do is to recursively dump the contents starting with the root node, and then display this information in the output `<div>`:

```
var html = dumpNode(rootNode);
document.getElementById('OutputDiv').innerHTML = html;
```

Admitted, that's cheating, since we did not implement the `dumpNode()` function yet. But it's not that hard: `dumpNode()` expects a node as its first argument. The function then notes the string representation of the node (for instance, `<Canvas>` has the string representation `Canvas`). If the node has a name, the name is remembered, as well:

```
function dumpNode(node) {
  var html = node.toString();
  if (node.name) {
    html += ' (' + node.name + ')';
  }
```

Now comes the tricky part. What if the node has child nodes? In that case, recursion comes into play: For every child node, `dumpNode()` calls `dumpNode()` again, so that every child node is dumped, as well. In order to visualize the hierarchy of the document, all child nodes are indented, using the `<blockquote>` HTML element.

There is one catch, however: If a node does not have children, accessing `node.children` will result in a JavaScript error. Therefore, we need to use a `try...catch` statement instead:

```
  try {
    if (node.children.count > 1) {
      html += '<blockquote>';
      for (var i = 0; i < node.children.count; i++) {
        html += dumpNode(node.children.getItem(i));
      }
      html += '</blockquote>';
    }
  } catch (ex) {
  }
```

Finally, the `dumpNode()` function adds a line break at the end and returns the HTML markup:

```
  html += '<br />';
  return html;
}
```

For security reasons, you may want to HTML escape the element names by getting rid of &, <, >, ', and ", replacing it with `&`, `<`, `>`, `&apos;`, and `&quot;` (in that specific order!).

Refer to Example 8-9 for the complete JavaScript code. Figure 8-4 shows the data dumped for a simple Hello World file.

Example 8-9. Dumping Silverlight content, the JavaScript file (SilverlightDumper.html.js)

```
function createSilverlight(XamlFile)
{
  Silverlight.createObjectEx({
    source: XamlFile,
    parentElement: document.getElementById('SilverlightPlugInHost'),
    id: 'SilverlightPlugIn',
    properties: {
      width: '600',
      height: '300',
      background:'#ffffffff',
      isWindowless: 'false',
      version: '1.0'
    },
    events: {
      onError: null,
    }
  });
}

function loadFile(f) {
  createSilverlight(f.elements['XamlFile'].value);
  document.getElementById('DumpButton').disabled = false;
}

function dump() {
  var plugin = document.getElementById('SilverlightPlugIn');
  var rootNode = plugin.content.root;
  var html = dumpNode(rootNode);
  document.getElementById('OutputDiv').innerHTML = html;
}

function dumpNode(node) {
  var html = node.toString();
  if (node.name) {
    html += ' (' + node.name + ')';
  }
  try {
    if (node.children.count > 1) {
      html += '<blockquote>';
      for (var i = 0; i < node.children.count; i++) {
        html += dumpNode(node.children.getItem(i));
      }
      html += '</blockquote>';
    }
```

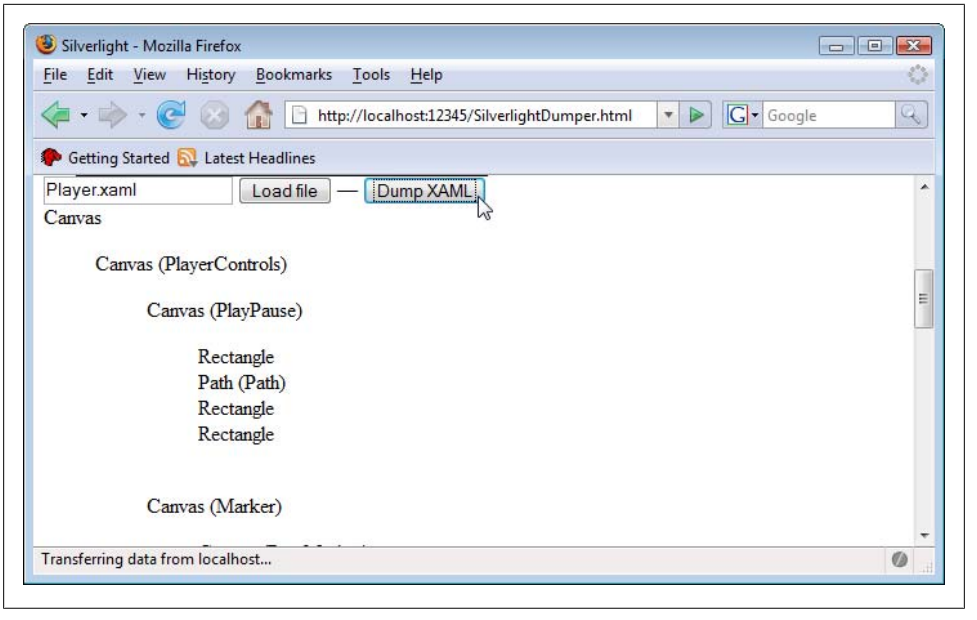


Figure 8-4. Dumping Silverlight data

```
        } catch (ex) {
        }
        html += '<br />';
        return html;
    }
```

There are many scenarios where combining HTML, JavaScript, and Silverlight makes sense. For instance remember that Silverlight does not offer a text input control yet, but HTML has that. So you could use HTML to enter data, and Silverlight to process it.

For Further Reading

JavaScript, The Definite Guide (http://www.oreilly.com/catalog/jscript5/index.html) by David Flanagan (O'Reilly)

The must-have book on JavaScript

CHAPTER 9

Special Silverlight JavaScript APIs

Advanced JavaScript APIs

This chapter is a mix of the Silverlight JavaScript capabilities introduced in the second part of this book and the JavaScript access to Silverlight content presented in Chapter 8. This chapter features some JavaScript APIs that are part of the Silverlight features but provide advanced possibilities. To be exact, Silverlight exposes a JavaScript API that lets developers download resources. While these resources are being downloaded, the download progress may be seen and you can process the downloaded data.

Dynamically Downloading Content

The technical background for Silverlight's downloader API is the `XMLHttpRequest` JavaScript object, which fuels everything Ajax. The Silverlight API does not copy the XMLHttpRequest API, but provides its own interfaces. You can trigger the API from both "XAML code-behind" and "HTML code-behind" JavaScript code, but generally you will want to start off from XAML.

When downloading content from within Silverlight, you usually have to follow these steps:

1. Initialize the downloader object by using the plugin's `createObject()` method
2. Add event listeners to the object to react upon important "happenings"
3. Configure the downloader object by providing information about the resource to download (HTTP verb, URL)
4. Send the HTTP request
5. Handle any occurring events

Let's go through these steps step by step. But first we need some XAML backing, so that we can wire up the whole code. Example 9-1 shows a suggestion.

Example 9-1. Downloading content, the XAML file (Download.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Loaded="loadFile">
  <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15" />
  <TextBlock x:Name="DownloadText" FontSize="52"
             Canvas.Left="25" Canvas.Top="35" Foreground="Black"
             Text="loading ..."/>
</Canvas>
```

Once the XAML has been loaded (`Loaded` event in the `<Canvas>` element), we start working on the download. Step 1 is creating the downloader object. For that, we first need to access the plugin (see Chapter 8 for details) and then call the method:

```
function loadFile(sender, eventArgs) {
  var plugin = sender.getHost();
  var d = plugin.createObject('downloader');
```

Step 2 is setting up event listeners. There are three events available:

`Completed`
: Download has been completed

`DownloadFailed`
: Download failed, for instance because the URL does not exist or there was an internal server error or security restrictions prevented the download

`DownloadProgressChanged`
: Another chunk of data has been downloaded from the server; there may be more to come

For this example, the `Completed` event is what we want:

```
d.addEventListener('completed', showText);
```

Step 3 is configuring the downloader object, which is done by using the `open()` method (this does not actually open the connection, but just configures the object). Provide the HTTP verb (usually `'GET'`), and the URI to request.

```
d.open('GET', 'Download.txt');
```

Step 4 is easy. The `send()` method does what exactly what it says—it sends the HTTP request.

```
d.send();
```

Finally, in step 5, you need to implement the event handlers; in our case, there's just one. The first argument for that event handler (we call it `sender`, as always) is conveniently the downloader object. It provides you with two options to access the data returned from the server:

`responseText`
: This property contains the response as a string

`getResponseText(id)`
: This method returns one specific part of a response. If you download a ZIP file, you can provide the file name of the file inside the archive that you want.

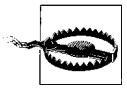
In this example we use the former option and output the text:

```
function showText(sender, eventArgs) {
  var textBlock = sender.findName('DownloadText');
  textBlock.text = sender.responseText;
}
```

Now all you need is a file called *Download.txt* (see the `open()` method call for the file name) residing in the same directory as your application, and you can download the file. After a (very) short while, you will see the content in the `<TextBlock>` element. Example 9-2 contains the complete "XAML code-behind" JavaScript code, and Figure 9-1 shows the output. If you are using a tracing tool like Firebug (see Figure 9-2), you will see that the text file is actually downloaded.

Example 9-2. Downloading content, the XAML JavaScript file (Download.xaml.js)

```
function loadFile(sender, eventArgs) {
  var plugin = sender.getHost();
  var d = plugin.createObject('downloader');
  d.addEventListener('completed', showText);
  d.open('GET', 'Download.txt');
  d.send();
}

function showText(sender, eventArgs) {
  var textBlock = sender.findName('DownloadText');
  textBlock.text = sender.responseText;
}
```



Downloading only works using the HTTP protocol, you cannot use the `file:` protocol (which means that you have to run these examples using a web server, even if it is a local one). You also have to adhere to the *same-origin policy* of the browser security system. The URL you are requesting must reside on the same domain, use the same port, and the same protocol.

It's a good idea to provide the user with information on how long downloading big files will take. The `DownloadProgressChanged` event is an excellent choice to implement this. We start off again with a simple XAML file (see Example 9-3) that both starts JavaScript code once the canvas has been loaded and includes a text block that will hold the progress information.

Example 9-3. Displaying the download progress, the XAML file (DownloadProgress.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Loaded="loadFile">
```

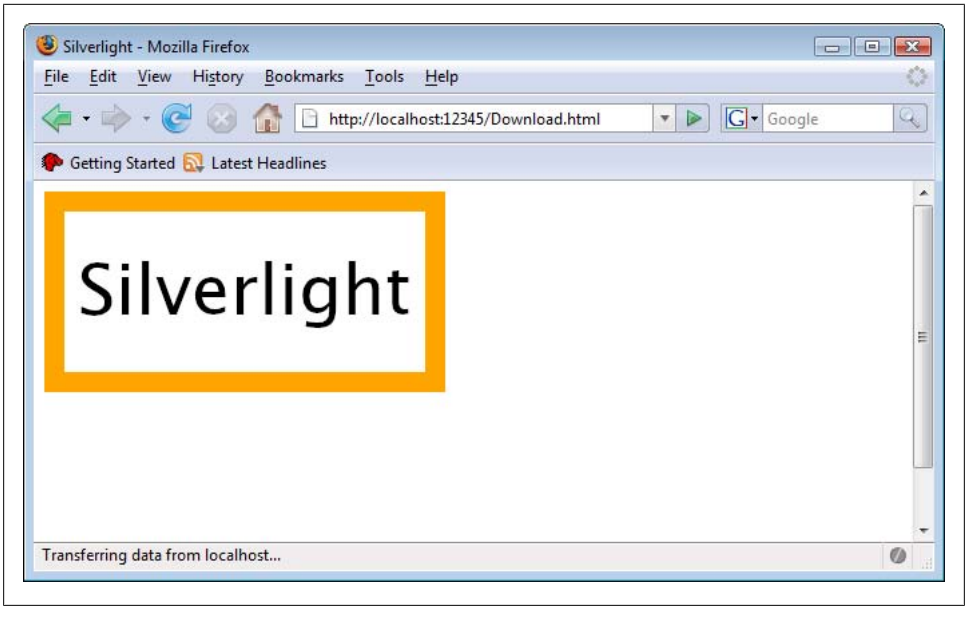


Figure 9-1. The content is downloaded

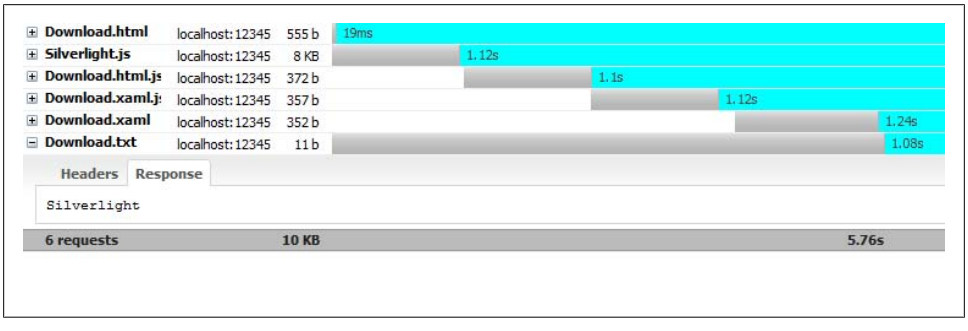


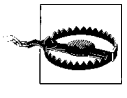
Figure 9-2. Firebug unveils the HTTP traffic, including the request to the text file

```
    <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15" />
    <TextBlock x:Name="ProgressMeter" FontSize="64"
               Canvas.Left="60" Canvas.Top="35" Foreground="Black"
               Text="0 %"/>
</Canvas>
```

The best way to simulate a big file is to write a script that does not return that much data, but takes some time to run. The ASP.NET code from Example 9-4 just sends a bit over 1,000 bytes, but takes a break every few dozen bytes. Therefore, the script is continuously sending data, but taking a few seconds to finish.

Example 9-4. Displaying the download progress, the ASP.NET file simulating a big or slow download (DownloadProgress.aspx)

```
<%@ Page Language="C#" %>

<script runat="server">
void Page_Load() {
  Response.Clear();
  Response.BufferOutput = false;
  Response.AddHeader("Content-Length", "1003");
  for (int i = 0; i < 17; i++)
  {
    Response.Write(
      "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA");
    System.Threading.Thread.Sleep(500);
  }
  Response.End();
}
</script>
```



It is very important that the script, in whatever language it is written, does not use output buffering and also sends the correct `Content-Length` HTTP header. Otherwise Silverlight cannot know how many bytes the server will send in its response, and thus cannot determine how many percent of the data have already been sent.

Now to the JavaScript part. The beginning is easy: create a downloader object and send an HTTP request. Only this time, we are listening to the `DownloadProgressChanged` event:

```
function loadFile(sender, eventArgs) {
  var plugin = sender.getHost();
  var d = plugin.createObject('downloader');
  d.addEventListener('DownloadProgressChanged', showProgress);
  d.open('GET', 'DownloadProgress.aspx');
  d.send();
}
```

The downloader object exposes a property called `downloadProgress`, which returns a value between 0 and 1, which is a percentage of how much data has already been transferred. Multiplied by 100 (and rounded), you get a nice percentage value that can be displayed in the `<TextBlock>` element. Without further ado: Example 9-5 contains the complete JavaScript code.

Example 9-5. Displaying the download progress, the XAML JavaScript file (DownloadProgress.xaml.js)

```
function loadFile(sender, eventArgs) {
  var plugin = sender.getHost();
  var d = plugin.createObject('downloader');
  d.addEventListener('DownloadProgressChanged', showProgress);
  d.open('GET', 'DownloadProgress.aspx');
  d.send();
}
```

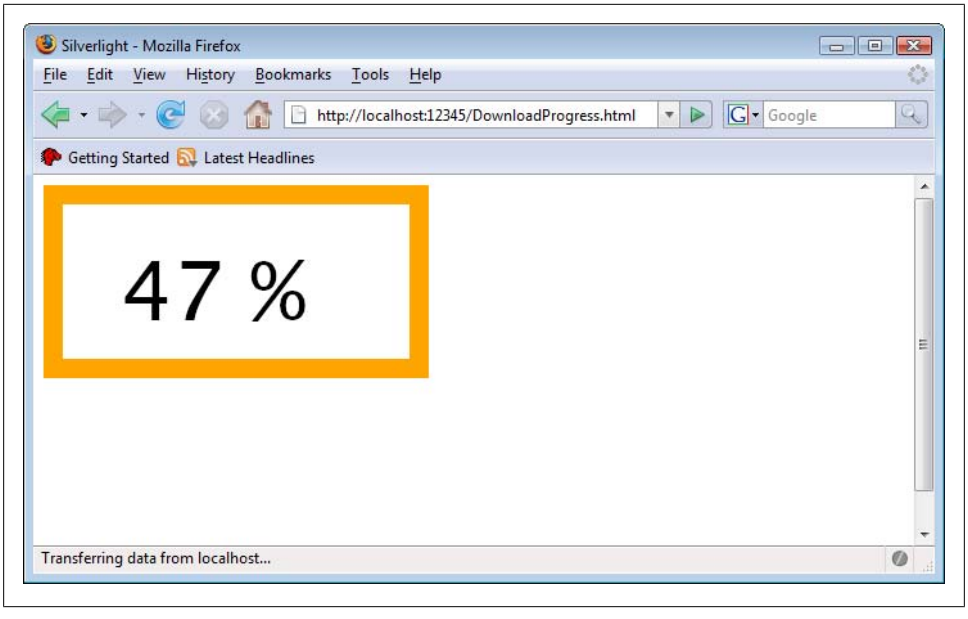


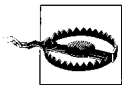
Figure 9-3. The loading file

```
function showProgress(sender, eventArgs) {
  var progress = sender.downloadProgress;
  var textBlock = sender.findName('ProgressMeter');
  textBlock.text = Math.round(100 * progress) + ' %';
}
```

Figure 9-3 shows the result—the file is loading, and loading, and loading.

Using Additional Fonts

One of the font support limitations mentioned in Chapter 4 is that there are only a handful of fonts supported, but at least they work on all supported browsers and operating systems. With the downloader object and some JavaScript code, additional fonts can be used, as long as they are TrueType (TTF) or OpenType fonts. Another limitations: they need to use the `.TTF` file extension.



Probably the biggest limitation of them all is a legal one: You have to make the TTF files available to Silverlight so that the application may download them. If the application can download the fonts, then any user can. So you need to make sure that you are allowed to distribute the fonts you are using. For this very reason, the book downloads do not come with the fonts mentioned in this chapter—you need to use your own. For registered Visual Studio 2005 users, there is a font download (see the "Further Reading section) that you might want to experiment with. (But again, you are not licensed to distribute this font to others.)

As usual, we start with a simple XAML file that will be enriched with JavaScript code. Example 9-6 contains a rectangle (for visual reasons) and a `<TextBlock>`. The text block uses the Lucida font by default (see Chapter 4). We want to change this using JavaScript, therefore we set up an event handler for the `Loaded` event so that we can load a font once the XAML has been loaded.

Example 9-6. Loading a font, the XAML file (TrueTypeFont.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Loaded="loadFont">
  <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15" />
  <TextBlock x:Name="FancyText" FontSize="64" Canvas.Left="20" Canvas.Top="35"
             Foreground="Black" Text="Silverlight" />
</Canvas>
```

Loading the font starts the same as loading any file using the downloader object, it makes an HTTP request and sets up a `Completed` event handler. Make sure that you have the `CALIBRI.TTF` (the Calibri font installed as part of Office 2007 and Windows Vista) file available or use another font file and change the code accordingly.

```
function loadFont(sender, eventArgs) {
  var plugin = sender.getHost();
  var d = plugin.createObject('downloader');
  d.addEventListener('Completed', displayFont);
  d.open('GET', 'CALIBRI.TTF');
  d.send();
}
```

In the event handler function, we cannot use the HTTP response as a string. However, the `<TextBlock>` object supports a method called `setFontSource()`. If you provide the downloader object as an argument, you can use the font you just downloaded!

```
function displayFont(sender, eventArgs) {
  var textBlock = sender.findName('FancyText');
  textBlock.setFontSource(sender);
```

Remember that the download object is automatically the event sender and, therefore, the first argument for any event handler function attached to the object.

The rest is easy: you can now set the `fontSize` property of the `<TextBlock>` element to the name of the downloaded font:

Figure 9-4. The text now uses the Calibri font, which is on all supported systems

```
    textBlock.fontFamily = 'Calibri';
}
```

Example 9-7 contains the complete code, and you can see the result in Figure 9-4.

Example 9-7. Loading a font, the XAML JavaScript file (TrueTypeFont.xaml.js)

```
function loadFont(sender, eventArgs) {
  var plugin = sender.getHost();
  var d = plugin.createObject('downloader');
  d.addEventListener('Completed', displayFont);
  d.open('GET', 'CALIBRI.TTF');
  d.send();
}

function displayFont(sender, eventArgs) {
  var textBlock = sender.findName('FancyText');
  textBlock.setFontSource(sender);
  textBlock.fontFamily = 'Calibri';
}
```

But that's not all! You can even download a ZIP file containing several font files, and use every font in there. The XAML files does not change from that in Example 9-6, except for one thing: clicking on the canvas executes an event handler. The application will change the currently displayed font whenever the user clicks on it. See Example 9-8 for the code.

Example 9-8. Loading multiple fonts, the XAML file (TrueTypeFonts.xaml)

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Loaded="loadFont" MouseLeftButtonDown="displayFont">
  <Rectangle Width="300" Height="150" Stroke="Orange" StrokeThickness="15" />
  <TextBlock x:Name="FancyText" FontSize="64" Canvas.Left="20" Canvas.Top="35"
             Foreground="Black" Text="Silverlight" />
</Canvas>
```

Now create a ZIP file containing several TTF files; we are using Consolas and Calibri here. Then load the ZIP file as usual, in the `loadFont()` function. The `displayFont()` function will now be called on two occasions: when the font ZIP file has been loaded and when the user clicks on the rectangle. In the former case, the font source of the text box needs to be set to the downloader object. This only works if the event sender (first argument of the event handler function) is the downloader object, and not the `<Canvas>` element (when the user clicks it). A `try...catch` block avoids any JavaScript errors:

```
function displayFont(sender, eventArgs) {
  var textBlock = sender.findName('FancyText');
  try {
    textBlock.setFontSource(sender);
  } catch (ex) {
  }
  //...
}
```

The rest is easy. Once the font source is set, you can set the `TextBlock` element's `fontFamily` property to the name of any font within the ZIP file. Example 9-9 shows the complete code, which also sets the correct font size depending on the font being used. You can see in Figure 9-5 that the application now also uses the Consolas font.

Example 9-9. Loading multiple fonts, the XAML JavaScript file (TrueTypeFonts.xaml.js)

```
var font = 'Calibri';

function loadFont(sender, eventArgs) {
  var plugin = sender.getHost();
  var d = plugin.createObject('downloader');
  d.addEventListener('Completed', displayFont);
  d.open('GET', 'fonts.zip');
  d.send();
}

function displayFont(sender, eventArgs) {
  var textBlock = sender.findName('FancyText');
  try {
    textBlock.setFontSource(sender);
  } catch (ex) {
  }
  if (font == 'Calibri') {
    font = 'Consolas';
    var fontSize = '40';
  } else {
    font = 'Calibri';
    var fontSize = '64';
  }
  textBlock.fontFamily = font;
  textBlock.fontSize = fontSize;
}
```

So the downloader class can not only be used to download text information, but also to provide dynamic resources like TrueType/OpenType fonts.

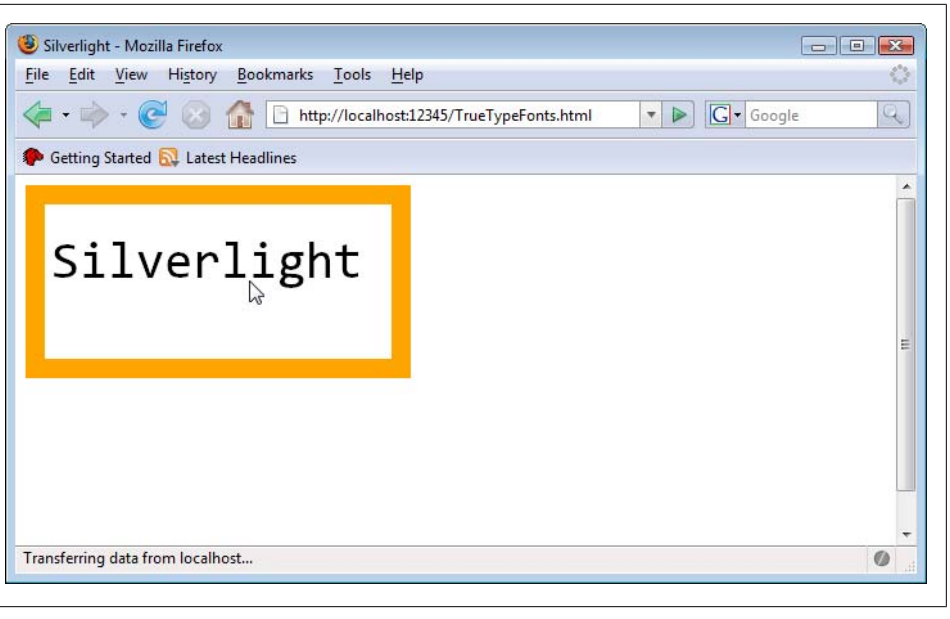


Figure 9-5. Clicking on the canvas changes the font

Loading Images

You can also use the downloader class to load images that are then loaded into an `<Image>` or `<ImageBrush>` element. The object method is called `setSource()` again, but this time expects two arguments:

- The downloader object
- The filename of the image to use, if you downloaded a ZIP with multiplied images inside. If you directly downloaded just one image, use an empty string as the second method argument.

When downloading big images, you can also track the download progress of those images.

Further Reading

http://www.microsoft.com/downloads/details.aspx?FamilyID=22e69ae4-7e40-4807-8a86-b3d36fab68d3&DisplayLang=en
: Consolas font (for registered Visual Studio 2005 users)

CHAPTER 10

ASP.NET 2.0, ASP.NET AJAX, and Silverlight

The ASP.NET Futures

The ASP.NET Futures is a collection of controls and features that did not make it in the release version of ASP.NET 2.0 but may be subject to further inclusion (or may be eternally doomed). It offers a sneak peek at what Microsoft is working on or planning, and is subject to frequent change. The ASP.NET Futures are released in form of Community Technology Previews (CTPs); this chapter is based on the July 2007 CTP.

Probably the best-known part of the ASP.NET Futures are the ASP.NET AJAX Futures, a set of add-ons for the ASP.NET AJAX Extensions. Most elements in the ASP.NET AJAX Futures are features that were part of pre-release versions of ASP.NET AJAX (then codenamed Atlas), but were removed from the core and shelved in the Futures project.

Beginning with the May 2007 CTP, the ASP.NET Futures included special ASP.NET 2.0 web controls that facilitate embedding Silverlight XAML and media content into ASP.NET pages. This chapter will help you set up the ASP.NET Futures on your development machine and then have a look at these two controls.

Installing the ASP.NET Futures

Before you can install the ASP.NET Futures, you need to get the ASP.NET AJAX Extensions first, since the Futures is an add-on to Microsoft's AJAX framework. You can go to Microsoft's ASP.NET AJAX homepage, *http://www.asp.net/ajax/*, to get the ASP.NET AJAX Extensions. The direct download link is *http://www.microsoft.com/downloads/details.aspx?FamilyID=ca9d90fa-e8c9-42e3-aa19-08e2c027f5d6&displaylang=en*.

The installer copies the ASP.NET AJAX files to your machine, including the server assembly (which will be installed in the Global Assembly Cache, or GAC), and the

client script libraries. It also integrates into Visual Studio and Visual Web Developer Express Edition, providing a new web site template. If you are using Visual Studio 2008, ASP.NET AJAX is already included in all .NET Framework 3.5 web sites, so you can skip this step.

The July CTP is available for download in the Microsoft download center at *http://www.microsoft.com/downloads/details.aspx?FamilyId=A5189BCB-EF81-4C12-9733-E294D13A58E6&displaylang=en*.

If there is a more recent release when you are reading this, there may be a link to the latest bits on that page. Another way to get to the newest release is to go to *http://asp.net/ajax/downloads/* and click on the "Download the Futures" link there. Since the ASP.NET AJAX Futures are part of the ASP.NET Futures, you will automatically get to the correct page. Note, however, that this chapter is based on the July CTP, so there is no guarantee that it will still work with following CTPs (but the odds are that it will continue to work).

The ASP.NET Futures installer (see Figure 10-1) copies all important files, including some documentation, to your machine. It also registers with Visual Studio 2005 (including the free Visual Web Developer Express Edition). When setting up a new web site, you have two new templates: *ASP.NET Futures Web Site* and *ASP.NET Futures AJAX Web Site* (see Figure 10-2). The latter is a web site with additional ASP.NET AJAX components, and the former is what we want.

Create a web site based on the *ASP.NET Futures Web Site* template and call it *SilverlightFutures* (you don't have to, but this project name is used throughout this chapter). The first thing you will notice is that the web site's *Bin* directory is filled with a bunch of DLLs (see Figure 10-3). These provide the additional functionality of the ASP.NET Futures, including the two controls we will cover in the following section.

Embedding XAML

One control provided by the ASP.NET Futures that is relevant to Silverlight is `<asp:Xaml>`. This control lets you embed XAML content into an ASP.NET page. We will have a look at what the client-side HTML output is in a second; but before that, here are the most important attributes of `<asp:Xaml>`:

`XamlURL`
: The URL of the Silverlight XAML file to include.

`MinimumSilverlightVersion`
: The minimum version of Silverlight that needs to be checked for. If you are displaying Silverlight 1.1 content, set this property to "1.1".

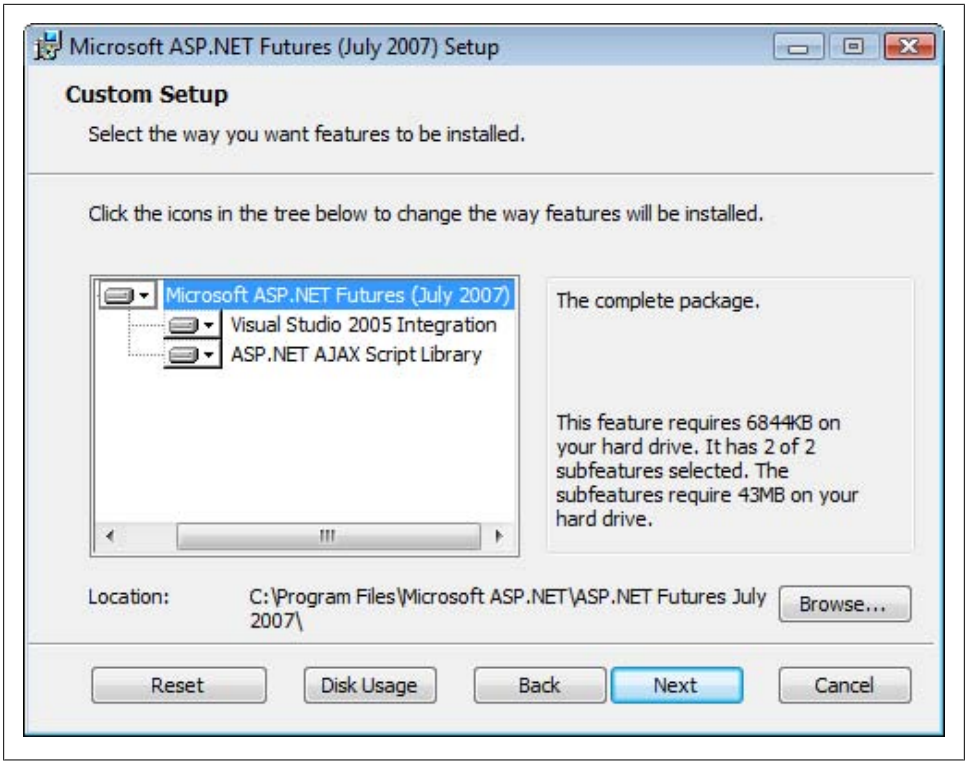


Figure 10-1. The ASP.NET Futures installer

`ScaleMode`
: How to scale the embedded media. Possible values are `ScaleMode.None` (no scaling), `ScaleMode.Stretch` (stretch without keeping the aspect ratio), and `ScaleMode.Zoom` (stretch and keep the aspect radio).

`Width`
: The width (in pixels) of the Silverlight control.

`Height`
: The height (in pixels) of the Silverlight control.

Here is an example:

```
<asp:Xaml ID="Xaml1" runat="server" XamlURL="Logo.xaml"
          MinimumSilverlightVersion="1.0" Width="250" Height="250" />
```

It is also possible to load JavaScript files that are used by the XAML content. This is done exactly like the ASP.NET AJAX `ScriptManager` web control does it, by using the `<Script>` sub element:

```
<asp:Xaml ID="Xaml1" runat="server" XamlURL="Logo.xaml"
          MinimumSilverlightVersion="1.0">
  <Scripts>
```

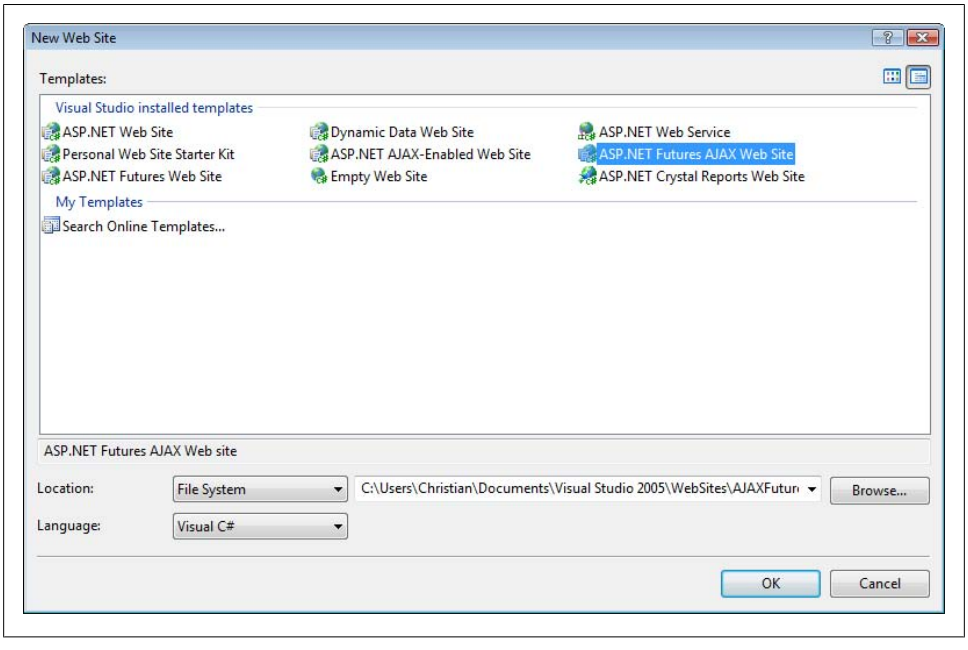


Figure 10-2. The ASP.NET Futures install new Visual Studio templates

```
    <asp:ScriptReference Path="myLibrary.js" />
  </Scripts>
</asp:Xaml>
```

Talking about the `ScriptManager`: This is the web control that comes with ASP.NET AJAX. It is responsible for setting up a web page to use ASP.NET AJAX effects. In order to use `<asp:Xaml>`, you do need the `ScriptManager` on the page:

```
<asp:ScriptManager id="ScriptManager1" runat="server" />
```

Example 10-1 contains the complete code:

Example 10-1. The XAML control (Xaml.aspx)

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Silverlight</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div>
      <asp:Xaml ID="Xaml1" runat="server" XamlUrl="Logo.xaml"
                MinimumSilverlightVersion="1.0" Width="250" Height="250">
```

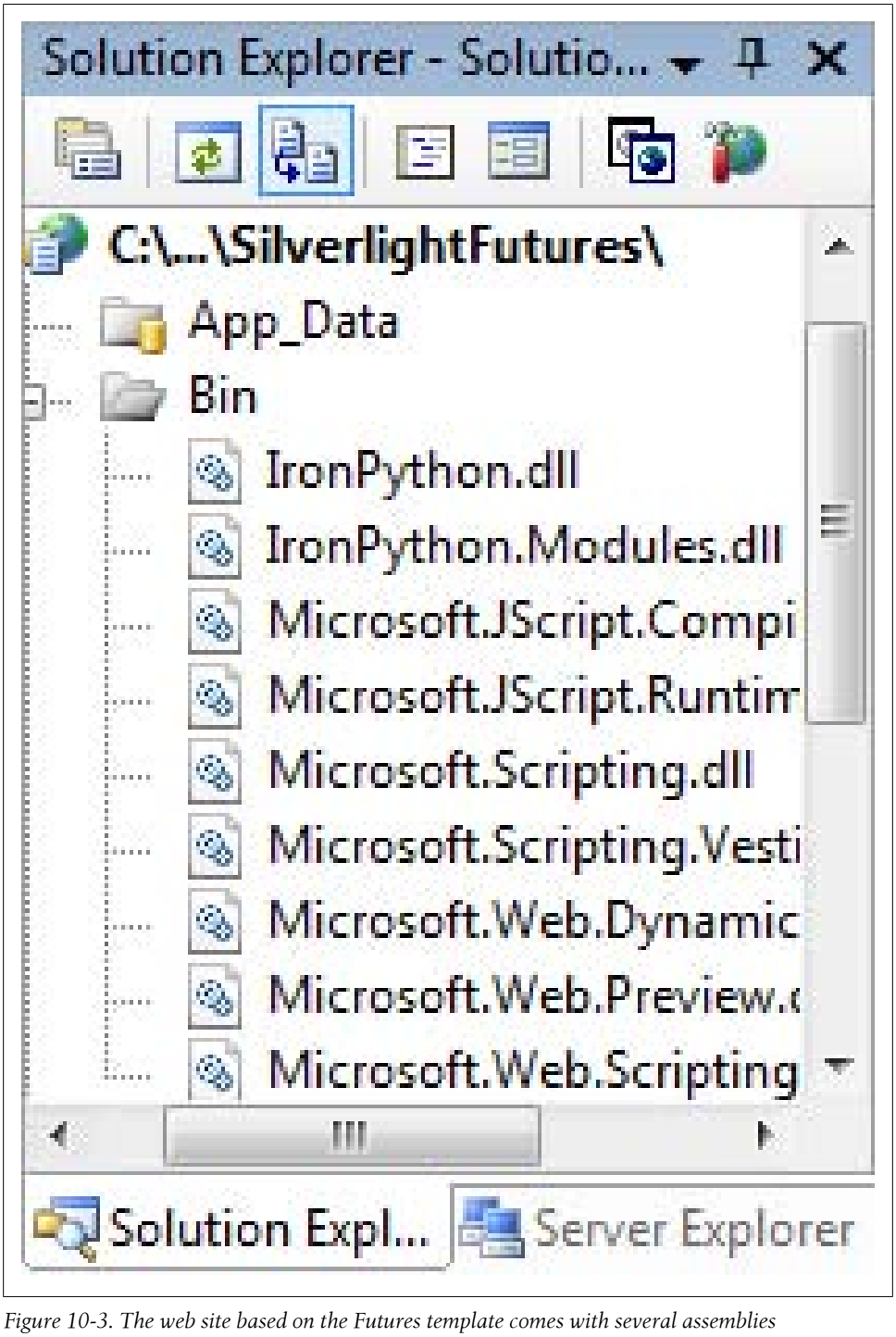


Figure 10-3. The web site based on the Futures template comes with several assemblies

```
        <Scripts>
          <asp:ScriptReference Path="myLibrary.js" />
        </Scripts>
      </asp:Xaml>
    </div>
  </form>
</body>
</html>
```

Let's have a look at the output of this code which reaches the client browser:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head><title>
  Silverlight
</title></head>
<body>
    <form name="form1" method="post" action="Xaml.aspx" id="form1">
<div>
<input type="hidden" name="__EVENTTARGET" id="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" id="__EVENTARGUMENT" value="" />
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUBMGRkCBGFCdJgCnCCrZD2vBQzbiHmKxE=" />
</div>

<script type="text/javascript">
<!--
var theForm = document.forms['form1'];
if (!theForm) {
    theForm = document.form1;
}
function __doPostBack(eventTarget, eventArgument) {
    if (!theForm.onsubmit || (theForm.onsubmit() != false)) {
        theForm.__EVENTTARGET.value = eventTarget;
        theForm.__EVENTARGUMENT.value = eventArgument;
        theForm.submit();
    }
}// -->

</script>


<script src="/SilverlightFutures/WebResource.axd?d=S-kP9O3jYyiEziOKSNtZnQ2&amp;
t=633197198135937500" type="text/javascript"></script>


<script src="/SilverlightFutures/ScriptResource.axd?d=DNOeWsX-g6VyC5Xz7xk7W
vgFhN4SHCwrIK-9ncFuYLMubPrRGYGCb2rOhd6nkacvx5L46hlWVnzQwttRkL7OPA9Mmwzt7BPN
wOJnraQhQxo1&amp;t=633223480818281250" type="text/javascript"></script>
<script src="/SilverlightFutures/ScriptResource.axd?d=DNOeWsX-g6VyC5Xz7xk7W
vgFhN4SHCwrIK-9ncFuYLMubPrRGYGCb2rOhd6nkacvx5L46hlWVnzQwttRkL7OPJuhioH-svy4
FH67UeRRAC8JoJigfq1BpYHExTBWbq960&amp;t=633223480818281250"
type="text/javascript"></script>
<script src="/SilverlightFutures/ScriptResource.axd?d=serzne1tmmxXTszmVvgl8
```

```
B8VX5vGO5GTdxLZnG1P9TxLuzI_yuYj39p1RZqIcJNt0&amp;t=633210654600000000"
type="text/javascript"></script>
<script src="myLibrary.js" type="text/javascript"></script>
    <script type="text/javascript">
//<![CDATA[
Sys.WebForms.PageRequestManager._initialize('ScriptManager1',
document.getElementById('form1'));
Sys.WebForms.PageRequestManager.getInstance()._updateControls([], [], [], 90);
//]]>

</script>

    <div>
        <object id="Xaml1" type="application/x-silverlight">

</object>
    </div>


<script type="text/javascript">
<!--
Sys.Application.initialize();
Sys.Application.add_init(function() {
    $create(Sys.Preview.UI.Xaml.Control, {"minimumSilverlightVersion":"1.0",
"xamlSource":"Logo.xaml","width":"250","height":"250"}, null, null,
$get("Xaml1"));
});
// -->
</script>
</form>
</body>
</html>
```

This is quite a lot of code (compare that to the markup we have in the *Xaml.aspx* file!), so we will break it down into individual parts. The hidden form fields are added by ASP.NET, so nothing new or surprising here. This also includes the first `<script>` element.

However, the next four `<script>` elements are automatically added by Silverlight. One accesses *WebResource.axd*, the next three access *ScriptResource.axd*. These <script> elements load JavaScript resources that are embedded in the ASP.NET Futures assemblies. This includes the ASP.NET AJAX client script libraries and also special JavaScript code used to check for the Silverlight plug-in and to initialize the Silverlight content on the page, similar to the code first introduced in Chapter 2.

The next `<script>` element loads *myLibrary.js*, the file we provided in the `<Scripts>` section of `<asp:Xaml>`, using `<asp:ScriptReference>`. The remaining `<script>` sections on the page initialize both ASP.NET AJAX and the XAML control on the page. The XAML control is embedded using the `<object>` HTML tag:

```
<object id="Xaml1" type="application/x-silverlight"></object>
```

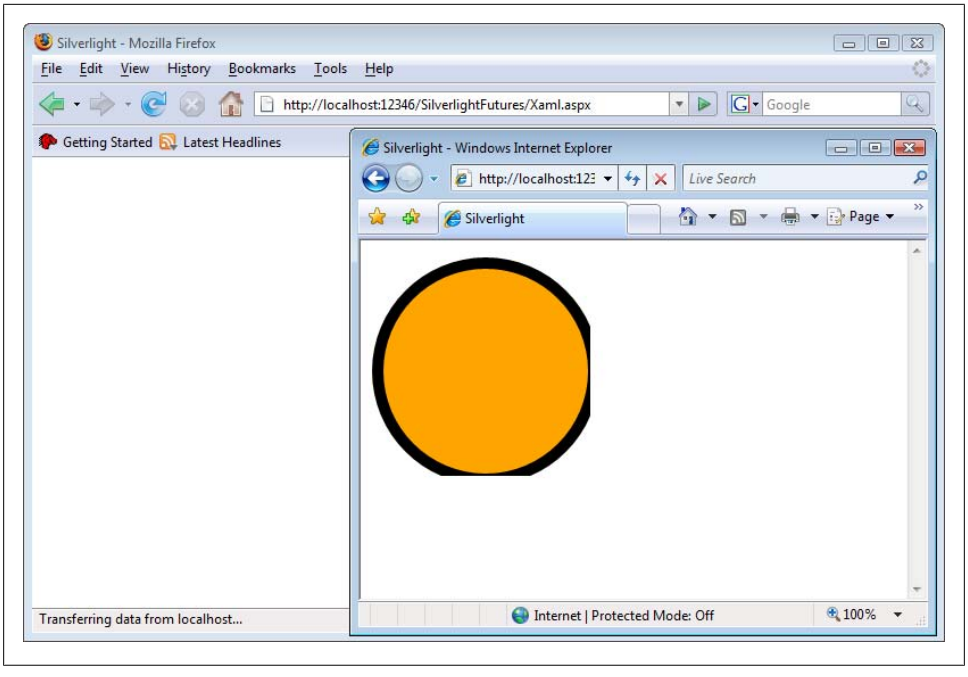


Figure 10-4. The browser output of the XAML control, if Height and Width are omitted

So, what we are basically getting here is similar output to the one created by the Silverlight project template. The XAML file is put into an `<object>` tag and a JavaScript library takes care of initializing the XAML content and checking for the correct plug-in. The markup created by `<asp:Xaml>` is a bit longer than our "hand-coded" markup, but it is also a bit faster to create in the IDE, because we don't have to write any JavaScript code ourselves anymore.

Make absolutely sure that you set the `Width` and `Height` properties, not all browsers figure that out automatically. Figure 10-4 shows what happens if you omit these two attributes. Internet Explorer shows the content (at least most of it), Firefox does not.

Embedding Media Content

If you only want to use media on your Silverlight page, i.e., movie and audio data, you can even avoid creating the necessary XAML yourself. The ASP.NET Futures contains a special web control called `<asp:Media>` that takes care of both setting up the required XAML and embedding it into the web page (like `<asp:Xaml>` does). The web control supports several properties, and the following is a list of the most interesting ones, apart from the mandatory `Width` and `Height`:

`MediaUrl`
: The URL of the video or audio file to embed

`AutoPlay`
: Whether the media shall be played automatically once the page has been loaded (`true`) or not (`false`)

`Muted`
: Whether the audio shall be off (probably no bad idea for commercial web sites)

`PlaceholderImageUrl`
: The URL of an image that is displayed as a placeholder while the media content is loaded

`ScaleMode`
: How to scale the content; choose from `ScaleMode.None` (no scaling), `ScaleMode.Stretch` (stretch without keeping the aspect ratio), and `ScaleMode.Zoom` (stretch and keep the aspect radio)

`Volume`
: The volume of the audio portion of the media

As with `<asp:Xaml>`, `<asp:Media>` also requires a `ScriptManager` control to be present. Example 10-2 shows a page with an embedded WMV video. No XAML or JavaScript is to be seen.

Example 10-2. Embedding media content (Media.aspx)

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Media.aspx.vb"
         Inherits="Media" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Silverlight</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div>
      <asp:Media ID="Media1" runat="server"
        MediaUrl="video.wmv" AutoPlay="true"
        Width="320" Height="240" />
    </div>
  </form>
</body>
</html>
```

The client output of the preceding code is very similar to the one `<asp:Xaml>` was creating in the previous section. However there are two obvious changes:

```
<div>
      <object id="Media1" type="application/x-silverlight">

</object>
    </div>
```

```
<script type="text/javascript">
<!--
Sys.Application.initialize();
Sys.Application.add_init(function() {
    $create(Sys.Preview.UI.Xaml.Media.Player, {"autoPlay":true,
"mediaUrl":"video.wmv","xamlSource":"/SilverlightFutures/WebResource.axd?d=
sIeXERCMlip1T8Xs8cc4lW872Ud9pErpWxrM2m2F_njre91sMOp-VS2I6MLtkdb1TXexyE7xlx9IdA
Iqyh66CyPBfuxgQSv7XtVtkFAxaM2-y2k5POy9hKqj4noBBzKiO&t=633210654600000000",
"width":"320","height","240"}, null, null, $get("Media1"));
});
// -->
</script>
```

So the media is embedded using the `<object>` HTML element. This is no surprise, since `<asp:Media>` is creating XAML markup to embed the video or audio content. However, the initialization of this `<object>` element, in the `<script>` section, is different. The `xamlSource` property is set to this URL (the exact name will vary on your system, because the current time is embedded in the URL):

```
/SilverlightFutures/WebResource.axd?d=sIeXERCMlip1T8Xs8cc4lW872Ud9pErpWxrM2m2F_
njre91sMOp-VS2I6MLtkdb1TXexyE7xlx9IdAIqyh66CyPBfuxgQSv7XtVtkFAxaM2-y2k5POy9hKqj
4noBBzKiO&t=633210654600000000
```

So there is no static XAML file that is sent down to the browser; instead the ASP.NET Futures creates a handler that will return dynamically generated XAML content. In case you are curious, use a tool like Firebug (*http://www.getfirebug.com/*) to sniff incoming traffic and have a look at the dynamic XAML. Figure 10-5 shows the result; as you can see, that's quite a lot of markup, since the output not only shows the media file, but also includes a visual user interface for it. How lucky we are that we did not have to create that ourselves.

The `Media` control also supports sub elements. One is `<Scripts>`, which allows you to load JavaScript libraries you need in that movie. Another one is `<Chapters>`. This allows you to use markup to add chapters to a movie (see Chapter 7 on how to do that manually). For each chapter marker, you need an `<asp:MediaChapter>` element and the following properties:

`TimeIndex`
: The start time of the chapter

`Title`
: The title of the chapter

`ImageUrl`
: The image that marks the chapter

Example 10-3 shows how this can look like, and Figure 10-6 the output in the web browser; when you hover over the video, all markers are shown. Clicking on one of them jumps to the appropriate position in the video.

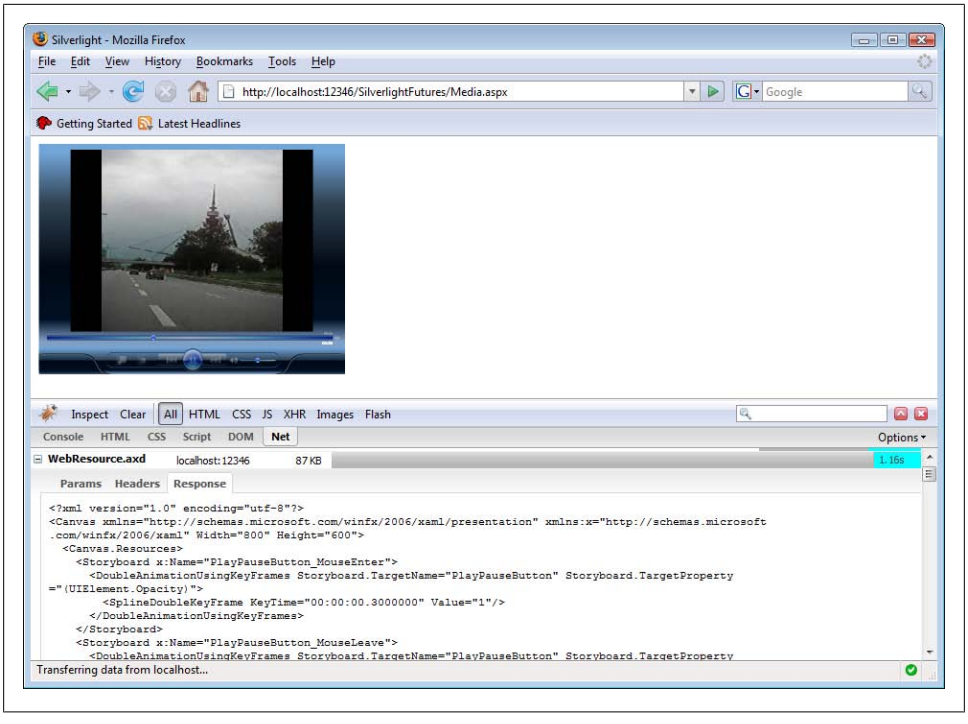


Figure 10-5. Firebug shows the XAML that was dynamically generated on the server

Example 10-3. Adding chapters to a video (MediaChapter.aspx)

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>Silverlight</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div>
      <asp:Media ID="Media1" runat="server"
        MediaUrl="video.wmv" AutoPlay="true"
        Width="320" Height="240">
        <Chapters>
          <asp:MediaChapter TimeIndex="13.200" Title="Junction"
                            ImageUrl="marker1.png" />
          <asp:MediaChapter TimeIndex="26.300" Title="Olympic Stadium"
                            ImageUrl="marker2.png" />
          <asp:MediaChapter TimeIndex="48.0" Title="Olympic Tower"
                            ImageUrl="marker3.png" />
        </Chapters>
      </asp:Media>
```

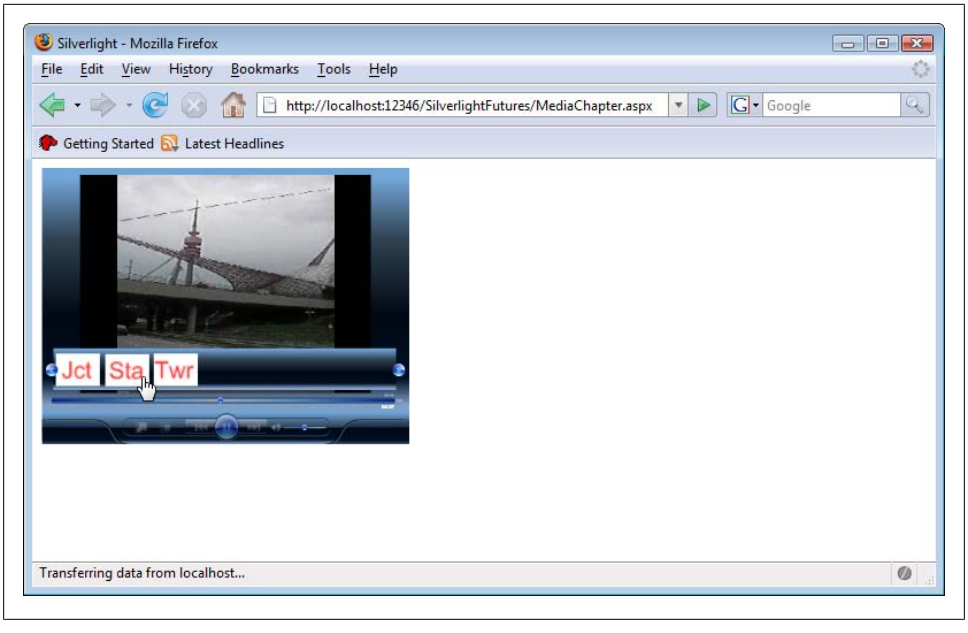


Figure 10-6. Chapters in a video

```
        </div>
      </form>
    </body>
    </html>
```

For Further Reading

Programming ASP.NET AJAX (http://www.oreilly.com/catalog/9780596514242/index.html) by Christian Wenz (O'Reilly)

All you need to know about ASP.NET AJAX and its related projects, including the ASP.NET AJAX Futures

CHAPTER 11

Silverlight 1.1 Preview

Silverlight's Future

This book is about Silverlight 1.0. The first release of the Silverlight technology contains quite a number of useful features, as the previous 10 chapters have shown. However, Microsoft is already working on future versions.

Unfortunately, the versioning system is really odd. The Silverlight 1.0 version comes as a small download with a nice but manageable feature set. The next version of Silverlight will add many additional feature, be a much bigger download, currently clocking in at about 4.5 MB on Windows (in contrast: Silverlight 1.0 is less than 1.4 MB). Mac OS X file sizes are bigger (for instance, Silverlight 1.1 currently comes in at almost 10 MB). So, you would expect that the new version number would be 1.5 at least, or maybe 2.0.

However, the next Silverlight version is called 1.1. This marginal increase of version number is not reflected in the feature set, which is vastly revamped and extended. It is also not clear what happens if there is a bugfix release or security update for Silverlight 1.0. Will the new version number be 1.0.1? Or, as Sun Microsystems is doing with Java, Silverlight 1.0 Update 1?

Apart from those irritating version numbering schemes, Silverlight 1.1 is really something to look forward to. This chapter touches very briefly on some of the upcoming features. As of time of writing, Silverlight 1.1 was still in alpha state, so investing too much time into the new version could be dangerous, as APIs can (and will!) change.

Currently, you can download both Silverlight versions (1.0 and 1.1) from the download page at *http://www.microsoft.com/silverlight/downloads.aspx*. The recommended development platform for Silverlight 1.1 content will be Visual Studio 2008; for its beta 2 release, Microsoft is providing add-ins that install templates and IntelliSense support at *http://www.microsoft.com/downloads/details.aspx?familyid=B52AEB39-1F10-49A6-85FC-A0A19CAC99AF&displaylang=en*.

Figure 11-1. Silverlight 1.1 must be present when you want to develop content for it

.NET Integration

JavaScript is, at least in my opinion, an underestimated language. It provides most features you need to enrich Silverlight content. However, to tell the truth, many developers ignored the language for a couple of years, at least until Ajax came up. And since Silverlight is targeted to WPF users, i.e., desktop developers, not all of them are familiar for JavaScript, since this language has been almost exclusively used for web applications.

A .NET developer's dream would be to use a .NET language in the Web–on the client side. Microsoft works hard to make this dream come true. Silverlight 1.1 comes with a stripped-down (but still very functional) version of the .NET Framework. That means that you can use a supported .NET language like C# or Visual Basic to create the "XAML code-behind" code. No need to use JavaScript any more! Of course, the browsers don't just start supporting .NET, the plug-in takes care of that. A special highlight is that this .NET support will also be available for the Mac OS X platform and a kind of .NET runtime will be included into the plug-in for the Apple operating system. This also means that Windows users do not have to have .NET Framework 3.0 or even 3.5 on their machines, everything that is required comes as part of the plug-in. (If not, the plug-in would probably not reach a high market share at all.)

Before you start developing content, you first need to install Silverlight 1.1 on your machine (see Figure 11-1). If the installer does not run, Visual Studio 2008 will not compile Silverlight content, since the compiler requires the Silverlight classes to be present.

We will demonstrate using a simple example. You will see that in the end, the code may be a bit more complex than before, but the possibilities are virtually endless. We will use Visual Studio 2008 (currently in Beta 2), and base the sample on the C# Sil-

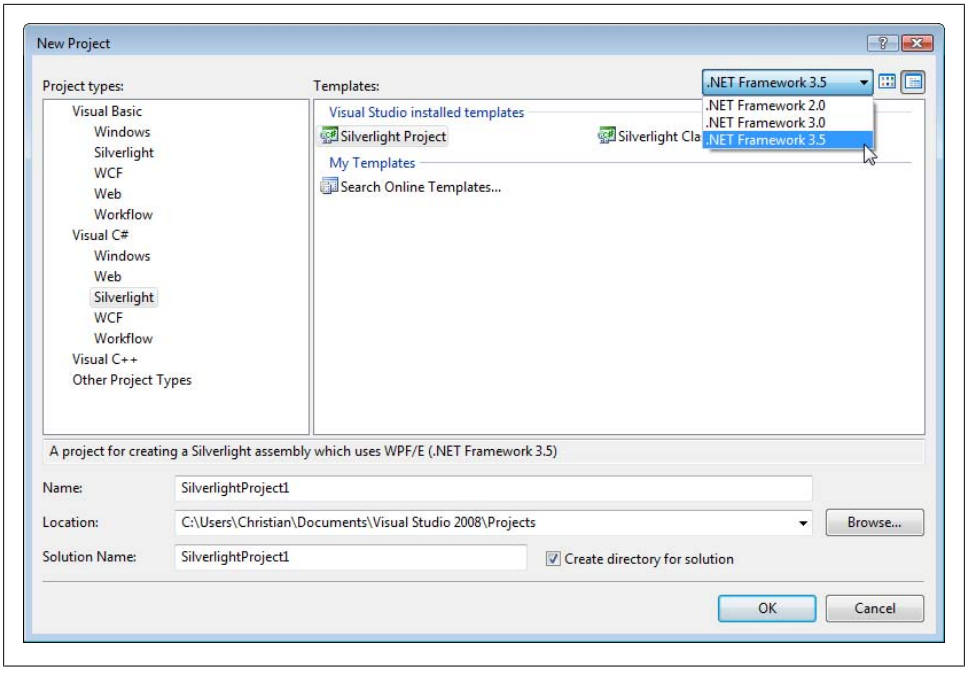


Figure 11-2. The Silverlight project templates in Visual Studio 2008

verlight project sample (see Figure 11-2). Make sure that you choose .NET Framework version 3.5, otherwise you will not see the project sample.

The project template installs these files:

Page.xaml
: A rudimentary XAML page.

Page.xaml.cs
: C# code-behind file, may be compared to the "XAML code-behind" JavaScript files we used throughout this book. If you use Visual Basic as the project language, the file will of course be called *Page.xaml.vb*.

Silverlight.js
: The well-known helper file used by the client.

TestPage.html
: A HTML test page prepared for displaying the content from *Page.xaml*.

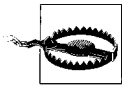
TestPage.html.js
: JavaScript code used by *TestPage.html* to load the XAML content from *Page.xaml*.

The sample application will output the current time in the Silverlight content (an advanced version of Hello World, so to speak). We will not need to touch the *.html* and *.html.js* pages but directly start with the XAML file. Example 11-1 shows the file after some elements were added.

Example 11-1. Using .NET in Silverlight, the XAML file (Page.xaml)

```
<Canvas x:Name="parentCanvas"
        xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Loaded="Page_Loaded"
        x:Class="Silverlight1._.Page;assembly=ClientBin/Silverlight1.1.dll"
        Width="640"
        Height="480"
        Background="White"
        >
  <Rectangle Width="450" Height="150" Stroke="Orange" StrokeThickness="15" />
  <TextBlock x:Name="MyText" FontFamily="Verdana" FontSize="56"
             Canvas.Left="30" Canvas.Top="40" Foreground="Black"
             Text="Silverlight" />
</Canvas>
```

Note the additional attributes the template has already included. Of special interest is `x:Class`, where the name of the code-behind class for the current XAML page and the file name of the associated project assembly is provided.



For this example, we have used the project name `Silverlight1.1`. In the current (alpha) version of the template, this confuses the template because there is a special character (the dot) in the project name. This dot is escaped in the class name, which is indeed `Silverlight1._.Page`. However the assembly name is, according to the template, *Silverlight.1._.dll*. As you can see in Example 11-1, this had to be changed to the actual name, *Silverlight.1.1.dll*, otherwise you'd get a JavaScript error message (if you see `AG_E_RUNTIME_MANAGED_ASSEMBLY_DOWNLOAD`, your assembly name is quite certainly wrong).

The code-behind file is now a real code-behind file, no JavaScript file that just happens to be called similarly to the XAML file. The `Loaded` event calls the `Page_Loaded()` method, which is now written in C#. As usual, it receives two arguments: the object that fired the event and event-specific arguments. First of all, the well-known `InitializeComponent()` method needs to be called, as usual in .NET:

```
public void Page_Loaded(object o, EventArgs e)
{
  // Required to initialize variables
  InitializeComponent();
```

We then need to access the `<TextBlock>` element on the page. Remember what we did with JavaScript in such a scenario? Indeed, we called the `findName()` method of the `<Canvas>` element, which happened to be the sender (first argument of the event handling method). This works with C# as well, we just have to take be careful about type casting because C# is strongly-typed. Another limitation is that C# is case-sensitive and uses camelcase, so we need to call `FindName()` instead of `findName()`.

```
Canvas can = o as Canvas;
TextBlock textBlock = (TextBlock)can.FindName("MyText");
```

Finally, the `Text` property (not `text`!) of the text block can be set. As a value we have access to most of what the .NET class library is offering, including `DateTime` values.

```
textBlock.Text = DateTime.Now.ToLongTimeString();
```

Example 11-2 contains the complete code for the XAML code-behind, and Figure 11-3 depicts the output.

Example 11-2. Using .NET in Silverlight, the XAML code-behind file (Page.xaml.cs)

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace Silverlight1._
{
    public partial class Page : Canvas
    {
        public void Page_Loaded(object o, EventArgs e)
        {
            // Required to initialize variables
            InitializeComponent();

            Canvas can = o as Canvas;
            TextBlock textBlock = (TextBlock)can.FindName("MyText");

            textBlock.Text = DateTime.Now.ToLongTimeString();
        }
    }
}
```

If you look at the HTML source code of this example, you will see that there are no surprises (which itself is no surprise, since the HTML file is static and only uses a bit of JavaScript). So the whole magic of running the .NET code is done by the Silverlight plug-in. If you have a look at the output of a tracing tool like Firebug (see Figure 11-3), you will see that the *.dll* assembly is loaded from the server and then processed.

Example 11-3. Using .NET in Silverlight, the HTML file (TestPage.html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <title>Silverlight</title>

    <script type="text/javascript" src="Silverlight.js"></script>
    <script type="text/javascript" src="TestPage.html.js"></script>
    <style type="text/css">
```

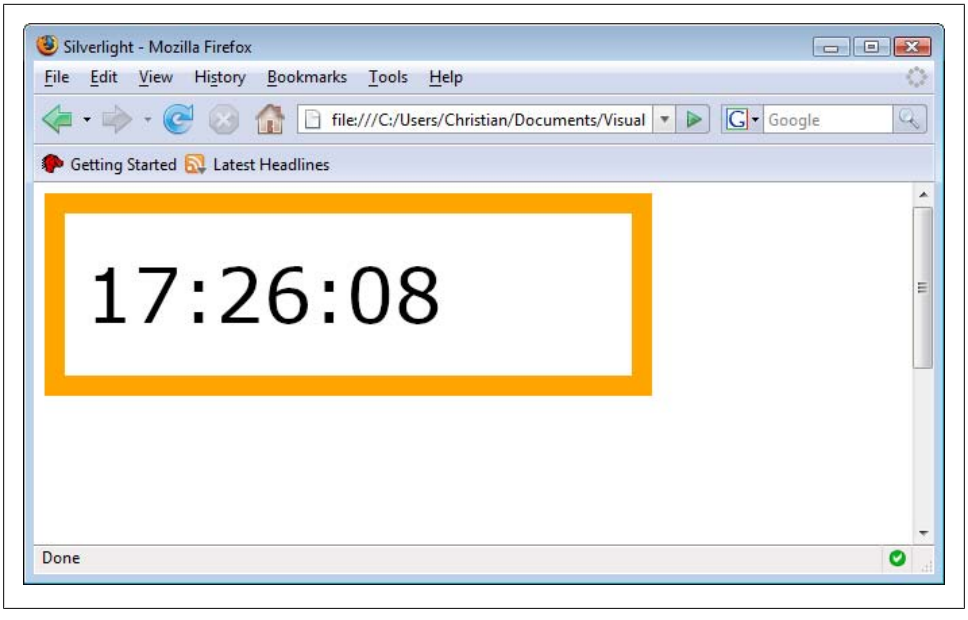


Figure 11-3. The current time appears in the Silverlight (1.1) content

```
            .silverlightHost { width: 640px; height: 480px; }
        </style>
    </head>

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

Further New Features

The .NET Framework subset supported by Silverlight 1.1 also includes some of the new additions in .NET Framework 3.5. One of the highlights is LINQ, a new way of querying data using an SQL-like syntax, but directly within the code (think SQL merging into C# and Visual Basic). Silverlight 1.1 will also come with a class library that contains additional features, like calling .NET Web Services. Further Silverlight 1.1 features include several additional controls, support for a new form of data web service Microsoft is currently working on (code-named Astoria), and much more. Make sure you visit *http://www.silverlight.net/* to stay up-to-date.

As Silverlight 1.1 evolves, this book will evolve, too; subsequent editions will focus on Silverlight 1.1. Microsoft hints that Silverlight 1.0 applications might still run with the

Silverlight 1.1 plug-in, but you never know what will happen. Therefore, this edition on the book focuses on what is there and (quite) stable. This chapters provided a sneak peek at the–possible–future.

Further Reading

http://www.microsoft.com/downloads/details.aspx?FamilyID=54b85d84-604d-43db-bcfe-7afd278208d8&DisplayLang=en
: The Silverlight 1.1 Alpha SDK

http://www.microsoft.com/downloads/details.aspx?FamilyID=811d8ad6-8d48-4684-b08c-686462d58a56&DisplayLang=en
: A print-quality reference poster showing the (current) Silverlight 1.1 architecture

http://www.microsoft.com/silverlight/downloads.aspx
: Microsoft Silverlight downloads

APPENDIX

Silverlight JavaScript Reference

Using the Silverlight Plug-in

There are two ways to access the Silverlight plug-in. Use the HTML DOM and the `document.getElementById()` method, and provide the ID used in the `createSilverlight()` function. Or intercept any Silverlight event and access the plug-in in the following fashion:

```
function eventHandler(sender, eventArgs) {
  var plugin = sender.getHost();
}
```

From the plug-in, you have access to the following properties and methods:

`content.accessibility`
: Accessibility information of the Silverlight content (read-only)

`content.actualHeight`
: The height of the Silverlight content (read-only)

`content.actualWidth`
: The width of the Silverlight content (read-only)

`content.createFromXaml(xamlContent, nameScope)`
: Creates a Silverlight object with the given XAML content; if `nameScope` is set to `true`, a unique `x:Name` attribute is assigned.

`content.createFromXamlDownloader(downloader, part)`
: Creates a Silverlight object from the given downloader object; if the downloaded content is a ZIP archive, `part` defines the name of the file to use

`content.findName(objectName)`
: Returns a reference to the object with the given name

`content.fullScreen`
: Whether the Silverlight plug-in is in full-screen mode (read and write)

`content.onFullScreenChange`
: Event handler that is fired when the application enters or leaves full-screen mode

`content.onResize`
: Event handler that is fired if the Silverlight content area is resized

`createObject(objectType)`
: Creates and returns an object with the given type

`initParams`
: The initialization parameters provided for the Silverlight content (read-only after the initialization)

`isLoaded`
: Whether the Silverlight plug-in and its XAML content are fully loaded (read-only)

`isVersionSupported(versionString)`
: Whether the current Silverlight plug-in supports content with the given version number

`onError`
: Event handler that is fired if an error occurs within the Silverlight application

`root`
: The root element of the Silverlight content (read-only)

`settings.background`
: The background color of the Silverlight content area (read and write)

`settings.enabledFramerateCounter`
: Whether the current frame rate is displayed in the browser's status bar or not (read and write; note that not all browser allow changing the content of the status bar)

`settings.enableRedrawRegions`
: Whether the regions of the Silverlight content that are redrawn are highlighted (read and write; only useful during development)

`settings.enableHtmlAccess`
: Whether to allow Silverlight content may access the HTML DOM (read-only)

`settings.maxFrameRate`
: The maximum frame rate (number of frames) of the Silverlight content (read and write)

`settings.windowless`
: Whether the Silverlight application runs in windowsless mode or not (then the background may use alphatransparency and can let the background on the HTML page shine through), set in the initialization phase (read-only afterwards)

`source`
: The XAML source code of the Silverlight content (read and write)