

# Introducing Silverlight 1.1

by Todd Anglin

Copyright © 2007 O'Reilly Media, Inc.

ISBN: 9780596515836

Released: October 2, 2007

*In this Short Cut, we'll take a close look at Silverlight 1.1 and bring clarity to this bleeding-edge technology. First we'll take a broad look at the new concept of Rich Internet Applications and how Silverlight fits into that picture. Then we'll examine how you work with Silverlight 1.1, highlighting tools, concepts, and code that you will need to build these interactive applications. Finally, we'll take a practical look at using Silverlight 1.1 to build a custom control, pulling all of the concepts and tools together.*

*Join ASP.NET Master and Telerik Chief Technical Evangelist Todd Anglin and the Telerik Team as they guide you to understanding Silverlight 1.1.*

## Contents

Understanding Silverlight 1.1 .....	2
What Is Silverlight 1.1? .....	7
An Overview of Rich Internet Applications .....	9
Working with Silverlight 1.1 .....	13
Building Applications with Silverlight 1.1 .....	60
Conclusions .....	75
Resources .....	75



## Understanding Silverlight 1.1

Even though Silverlight 1.0 has now been released, many developers are awaiting the arrival of Silverlight 1.1 before taking a serious look at this new technology. Silverlight 1.1 will add the .NET dimension to Microsoft's new Rich Internet Application platform, enabling the global community of .NET developers to apply their VB, C#, and other CLR language skills to writing Silverlight applications. But there's no need to wait; you can take Silverlight for a test drive today.

Before we can begin to understand Silverlight 1.1, we first need to understand Silverlight in general. As Microsoft succinctly puts it, "Silverlight is a cross-browser, cross-platform plug-in developed by Microsoft for delivering the next generation of .NET-based media experiences and Rich Internet Applications (RIAs) for the Web." But where did it come from and how does it compare to other new technologies like WPF? In this section, we'll review the history of the Silverlight project and compare Silverlight 1.1 to Silverlight 1.0 and WPF to gain a clear understanding of this new rich client framework.

### The Short History of Silverlight

Microsoft unveiled Silverlight for the first time in late 2005 at PDC 2005. At the time, it was introduced as Windows Presentation Foundation Everywhere (WPF/E), a subset of WPF technology that would work across platforms, delivered by the browser.

Almost a year later Microsoft released the first Silverlight CTP to the public. The December 2006 CTP version of Silverlight was very basic, enabling only JavaScript and XAML declarations to develop Silverlight applications. The February CTP improved upon the December release and for the first time started to attract serious attention from the Microsoft developer community. It added many new features, such as keyboard input (KeyUp/KeyDown), mouse cursor support, simple text metrics, and a unique full-screen mode. The February CTP in many ways kicked off the Silverlight buzz, with tutorials, articles, and even third-party frameworks all building on this early preview.

Despite the new popularity, many developers were still skeptical that Silverlight had any real potential. With only simple support for JavaScript and XAML, developers viewed Silverlight as a basic solution for media applications on the Web and much too weak to challenge perennial cross-browser plug-ins like Flash. Then came MIX07.

On April 30<sup>th</sup>, at MIX07 in Las Vegas, Microsoft announced *two* new versions of its Silverlight technology (in addition to the new "Silverlight" name): Silverlight 1.0 beta and Silverlight 1.1 alpha. The beta was an expected update to the February

CTP that had been around for a while, but the real excitement revolved around the 1.1 alpha announcement. Microsoft revealed that the 1.1 alpha version of Silverlight would enable Silverlight development using the .NET Framework, and it would sport a flexible programming model that supports C#, VB, AJAX ASP.NET, and many other CLR languages. Finally, the future of Silverlight was revealed, and .NET developers truly began to recognize the power of programming client-side applications with .NET.

In addition to the excitement generated by the .NET announcement at MIX, the official "Silverlight" name shocked many people after a series of clever Microsoft code names (like Avalon and Indigo) succumbed to W\*F product names by their release (such as Windows Communications Foundation, Windows Workflow Foundation, and Windows Presentation Foundation). For developers, Silverlight was a welcome change to the bland names that seemed to be dominating Microsoft development products. Furthering the cleverness of the name, all client files in Silverlight have the "AG" prefix, which is the atomic symbol for silver.

In the time that has passed since MIX, Microsoft has continued to develop and promote Silverlight. Silverlight 1.0, now available as an official release, has changed little since the MIX conference, and Silverlight 1.1 has definitely stolen much of the Silverlight press. Unfortunately, at the time of this writing Microsoft has yet to finalize the features, available .NET classes, or release date for Silverlight 1.1. For now, all Silverlight 1.1 work is being done on refreshed alpha bits, but a beta will likely be available around the end of the year or early 2008.

## Comparing Silverlight 1.0 and Silverlight 1.1

There are two Silverlight versions (so far), the first of which was recently released in September 2007:

- *Silverlight 1.0*: The freshmen Silverlight version is limited to running JavaScript only in the browser and it lacks any built-in UI controls. To achieve data input with Silverlight 1.0, the browser's native UI controls must be overlaid on top of the Silverlight Canvas. Silverlight 1.0 also lacks any concept of a control model, but some third-party developers have built frameworks on top of Silverlight 1.0 that reduce this shortcoming. Support for data formats is also limited, with native support for POX (plain old XML) and JSON only. Silverlight 1.0 is focused on delivering rich media experiences and paving the way for widespread distribution of the auto-updating Silverlight plug-in.
- *Silverlight 1.1*: The second iteration of Silverlight will be very different from its predecessor. It will offer a flexible programming model that supports C#, VB, IronPython, and IronRuby (among other languages). The added support

for .NET programming in Silverlight 1.1 will make it much easier for existing .NET developers to write Silverlight applications, and it will make it easier to integrate Silverlight with existing web applications. At the time of this writing, Microsoft has not announced when Silverlight 1.1 will ship.

Both Silverlight versions will have a number of similarities, including:

- Fast media capabilities thanks to a powerful new rendering engine (originally codenamed "Jolt"). This engine will enable high-quality audio and video to be easily delivered to all major browsers including Internet Explorer, Apple Safari, and Mozilla Firefox running on Mac OS X or Microsoft Windows.
- Silverlight supports playback of VC-1 video content across all supported browsers without requiring the Windows Media Player ActiveX control or Windows Media browser plug-ins.
- Silverlight provides a graphics system similar to WPF that integrates multimedia, graphics, animations, and interactivity into a single runtime. XAML, or Extensible Application Markup Language, can be used to create the vector graphics and animations used in Silverlight apps.

The Features Matrix below ([Table 1](#)) provides an overview of which runtime is required to use specific features in Silverlight applications.

**Table 1. Silverlight Runtimes Feature Matrix**

Features	Silverlight 1.0 Beta	Silverlight 1.1 Alpha
2D Vector Animation/Graphics	●	●
AJAX Support	●	●
Cross-Browser (Firefox, IE, Safari)	●	●
Cross-Platform (Windows, Mac)	●	●
Framework Languages (VB, C#, IronRuby, IronPython)		●
HTML DOM Integration	●	●
HTTP Networking	●	●
Isolated Storage		●
JavaScript Support	●	●
JSON Web Services		●

<b>Features</b>	<b>Silverlight 1.0 Beta</b>	<b>Silverlight 1.1 Alpha</b>
LINQ to Objects		●
Managed Control Framework		●
Managed HTML Bridge		●
Managed Exception Handling		●
Media – Content Protection		●
Media – 720P High Definition (HD) Video	●	●
Media – Audio/Video Support (VC-1, WMV, WMA, MP3)	●	●
Media – Image Support (JPG, PNG)	●	●
Media Markers	●	●
Rich Core Framework (e.g. Generics, etc.)		●
Security Enforcement		●
Silverlight ASP.NET Controls (asp:media, asp:xaml)	●	●
Type Safety Verification		●
Windows Media Server Support	●	●
XAML Parser (based on WPF)	●	●
XMLReader/Writer		●

## Comparing Silverlight 1.1 and WPF

Silverlight, formerly WPF/Everywhere, has a close relationship with the WPF framework. These technologies share a few key ingredients, but their target platform and scope are very different. WPF is a mature, fully scaled rendering and presentation technology targeted at the desktop. It allows a lot of computing-intensive UI concepts, like 3D rendering, and therefore depends on hardware acceleration for fast execution. Silverlight, on the other hand, is a young alpha technology (which means it is not feature complete) targeted at the browser. Like WPF, it utilizes XAML to create its UI markup, but the power of its rendering engine is considerably less than the full desktop WPF.

The differences between WPF and Silverlight can best be understood in the following comparison chart ([Table 2](#)):

**Table 2. Silverlight 1.1 compared to WPF**

<b>Silverlight</b>	<b>WPF</b>
Works on different operating systems – Windows, Mac OS, and Linux.	Requires Windows XP SP2, Windows Server 2003 SP1, or Window Vista.
Runs inside a hosted process. In other words, it must run inside of a browser, which means the technology targets the Web.	You can develop fully scaled WPF applications that run independently from a browser. You can also remotely distribute and execute a WPF application inside of a browser via XBAP (XAML Browser Applications).
Web oriented by nature, not targeted for development of offline applications.	Targeted primarily at Windows desktop applications.
Does not require .NET to be installed on the client.	Requires .NET 3.0 or later to be installed on the client.
The managed code runtime is implemented as a plug-in for different browsers (similar to Flash technology).	Implemented as a part of .NET 3.0 or later.
Supports multiple web browsers: <ul style="list-style-type: none"> <li>• Internet Explorer and Firefox on Windows; no support currently for Safari on Windows</li> <li>• Firefox and Safari on Mac OS X</li> </ul> Windows Vista and Windows XP Service Pack 2, Microsoft Internet Explorer 6, Windows Internet Explorer 7, Firefox 1.5.0.8 and 2.0.x, Apple Mac OS X, Firefox 1.5.0.8 and 2.0.x, Apple Safari 2.0.4.	Supports XBAP (XAML Browser Applications) execution only on Windows (currently MS .NET runs only on Windows) and requires Internet Explorer 6 or later.  NET 2.0: Windows 2000 Service Pack 3, Windows XP Service Pack 2, Windows Vista; .NET 3.0: Windows Vista and Windows XP Service Pack 2; Internet Explorer 6, Windows Internet Explorer 7.
Supports media playback natively as part of the technology.	Requires Windows Media Player 10 or later for media playback.
Supports a subset of XAML. There are serious limits in the alpha release, such as the lack of resources.	Supports full XAML specification.
Supports JavaScript and managed languages like C# and VB.NET. Also supports managed versions of scripting languages	Supports only managed languages (C#, VB, J#, etc.)



<b>Silverlight</b>	<b>WPF</b>
(through the Dynamic Language Runtime or DLR) like managed JavaScript, IronRuby, and IronPython.	
Prerequisites (the browser plug-in) size is around 6 MB for the Windows platform and around 10 MB for Mac.	Prerequisites size (.NET 3.0 or later, etc.) is more than 50MB.
Does not render 3D graphics.	Incorporates 3D technology.
Release date TBA; likely 2008.	Released in November of 2006.

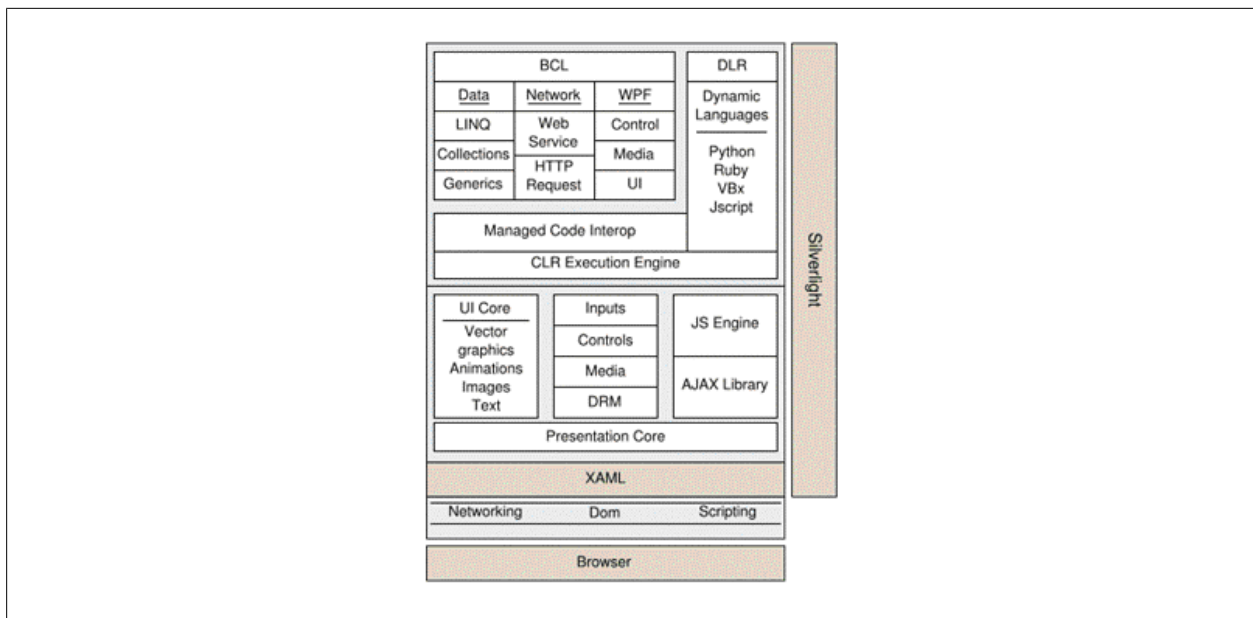
On a final note regarding WPF and Silverlight, it is a common misconception among .NET developers that Silverlight will enable them to directly place their WPF applications on the Web. In reality, the lightweight implementation of both the XAML rendering engine and .NET in Silverlight will make this a challenge. You will be able to share some of the XAML markup between your WPF and Silverlight applications, but porting a WPF application to the Web on Silverlight will not be an automatic or quick process.

## What Is Silverlight 1.1?

Silverlight 1.1 is Microsoft's second planned release of the Silverlight platform. Silverlight 1.1's biggest change from Silverlight 1.0 is the inclusion of a compact version of the .NET Framework, complete with the .NET Framework 3.0 Common Language Runtime. By adding .NET to Silverlight, Microsoft will make it easy for developers to reuse their existing programming skills to work with designers and quickly create rich applications for the Web.

And even though Silverlight 1.1 will bring .NET to the client, it can be easily integrated with many existing web technologies and backend web platforms. That means Silverlight will integrate with your existing infrastructure and applications, from IIS and .NET to Apache and PHP to simple JavaScript and XHTML on the client. Silverlight will not be a tool exclusive to ASP.NET web sites, which should result in broader adoption of the new technology.

Still, one of the key benefits of Silverlight 1.1 is that it can execute any .NET language, including C# and VB.NET. Unlike the CLR included with the "normal" .NET Framework, multiple instances of the core "Silverlight CLR" can be hosted in a single process. With this, the layout markup in the Silverlight XAML (.xaml) file can be augmented by code-behind code with all programming logic written in any .NET language.



**Figure 1. Silverlight framework model**

Silverlight 1.1 ships with a "lightweight" version of the full .NET Framework, which features—among other classes—extensible controls, XML Web Services, networking components, and LINQ APIs. This class library is a subset of (and is considerably smaller than) the .NET Framework's Base Class Library, which enables the Silverlight plug-in to be a fast and small download. For security, all Silverlight code runs in a sandbox environment that prevents invoking platform APIs, protecting user computers from malicious code. Silverlight 1.1 also adds support for DRM in media files, a fact that will make some people happy and others cringe.

In addition to the .NET Framework classes, Silverlight 1.1 also ships with a subset of the WPF UI programming model, including support for shapes, documents, media, and WPF animation objects. Silverlight 1.1 alpha does not ship with many WPF UI controls, though, so out-of-the-box controls are still very limited. Furthermore, in the current Silverlight 1.1 builds, the UI controls do not have any ability to bind to data. Microsoft says that the data binding limitations are strictly temporary and future builds of Silverlight 1.1 will eliminate the problem. Count on future Silverlight releases adding more UI controls, data binding support, and a much-needed automated layout manager to the Silverlight mix.

**Figure 1** illustrates the Silverlight framework model.



## An Overview of Rich Internet Applications

We have mentioned "Rich Internet Applications" a few times so far in the context of Silverlight, but what exactly is a "RIA" web application? And why would you want to adopt the RIA model for your own web development?

RIAs are web applications that have the features and functionality of traditional desktop applications. RIAs typically transfer the processing necessary for the user interface to the web client but keep the bulk of the data processing (such as maintaining the state of the program, the data, etc.) on the application server.

Traditional web applications center all activity on a client-server architecture, where a thin client (the web browser) interacts with a powerful server. Under this system, all processing is done on the server and the client is used only to display static HTML content. The biggest drawback with this system is that all interaction with the application must pass through the server. That means data must be sent to the server, the server must respond, and *then* the page must be reloaded on the client with the server's response. By moving more of this processing to client-side technology that can execute instructions on the client's computer, RIAs can circumvent this slow, synchronous loop for many user interactions.

### Benefits of Rich Internet Applications

One of the primary benefits of Rich Internet Applications is that they can offer user-interface behaviors that are not obtainable using only the HTML controls available in standard browser-based web applications. With a RIA platform, web applications are no longer limited by what the browser can do. Rather, they can implement any user interaction that the new RIA platform can support, such as drag-and-drop behaviors, smooth animations, and client-side calculations. While some of these interactions are possible without a RIA platform, the RIA approach is typically much more responsive and consistent across platforms.

The benefits of RIAs are not strictly looks, though. Using a client engine can also produce other performance benefits:

- *Client/server balance.* RIAs shift the balance of computing resources for web applications from the server to the client. This frees up resources on the web server, enabling the same server hardware to handle more concurrent user sessions. On the flip side, it requires that your users have computers that are powerful enough to execute complex client-side code, which generally is not a problem in this day and age.
- *Asynchronous communication.* The RIA client engine can interact with the server *asynchronously*—that is, without waiting for the user to perform an in-

terface action like clicking on a button or link. This feature enables RIA designers to move data between the user's PC and the server without making the user wait for the transfer to finish, similar to what Ajax is doing on the Web today.

- *Network efficiency.* Network traffic may also be significantly reduced in a RIA because an application-specific client engine can be more intelligent than a standard web browser when deciding what data needs to be exchanged with servers. Transferring less data for each interaction can speed up individual requests and responses, in turn reducing overall network load. Use of asynchronous *prefetching* techniques, however, can neutralize or even reverse this potential benefit. Because code cannot anticipate exactly what every user will do next, pre-fetching extra data, not all of which is actually needed by many users, is common.

### Shortcomings of Rich Internet Applications

While Rich Internet Applications offer some compelling advantages over current approaches to web development, there are a number of drawbacks that plague the technology—not the least of which is the requirement for a browser plug-in to function (in most cases). Among the more serious drawbacks of RIAs are:

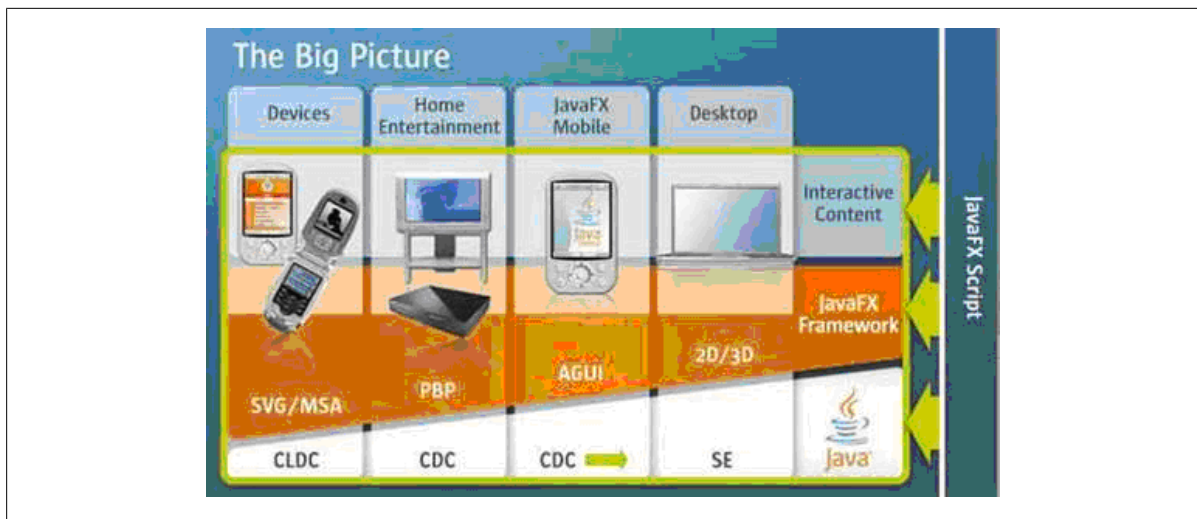
- *Sandbox.* Because RIAs run within a sandbox, they have restricted access to system resources. If users modify their systems or have reduced permissions that alter a RIA's ability to access system resources, RIAs may fail to operate correctly.
- *Disabled scripting.* RIAs usually require JavaScript or another scripting language to operate on the client. If the user has disabled active scripting in his browser, the RIA may not function properly, if at all.
- *Script download time.* Although it does not always have to be *installed*, the additional client-side intelligence (or *client engine*) of RIA applications needs to be delivered by the server to the client. While much of this is usually automatically cached, it needs to be transferred at least once. Depending on the size and type of delivery, client engine download time may be unpleasantly long, especially for users with slower Internet connections. Some RIA developers can lessen the impact of this delay by compressing scripts and by staging delivery over multiple pages of an application. For client engines that require a plug-in to be installed, this is not an option.

- *Loss of visibility to search engines.* Search engines may not be able to index the text content of RIA applications. This can be a major problem for web applications that depend on search engine visibility for their success.
- *Dependence on an Internet connection.* While the ideal network-enabled replacement for a desktop application would allow users to be "occasionally connected," wandering in and out of hotspots or from office to office, today the typical RIA requires network connectivity.

## RIA Technologies

Microsoft's Silverlight is certainly one of the newest entries in the RIA space, but it is not the only show in town. There are a number of platforms on the market that enable developers to create RIA applications, including the perennial favorite, Adobe Flash. Some of the top RIA platforms available today are:

- *Microsoft Silverlight:* Plug-in-based Microsoft technology is going head-to-head with other browser plug-in RIAs like Adobe Flash Player. Silverlight provides a rich graphics and animation display via XAML integration, and it also includes support for multimedia and HTML interaction. With version 1.1, the .NET framework will be included, enabling client-side programming with managed languages like C#. Like Adobe Flash Player, Silverlight code runs in a sandbox without direct access to platform APIs for security.
- *Adobe Flash Player and Adobe Flex:* Another way to build Rich Internet Applications—and probably the most popular to date—is Adobe Flash Player. These technologies are cross-platform and quite powerful for creating rich client-side interactions. Adobe Flex provides the option to create Flash UIs by compiling MXML, an XML-based interface description language. But perhaps the largest advantage of the Flash RIA platform is the install base of the plug-in, which at last count is installed on 98 percent of the world's computers.
- *AJAX Frameworks:* Ajax, or *Asynchronous JavaScript and XML* (or *XmlHttpRequest*), is a script-based RIA approach. JavaScript is the programming language with which Ajax calls are made, and the `XmlHttpRequest` browser object enables the asynchronous communication with the server. Data retrieved using this technique is commonly formatted using XML, though it is not a requirement. Ajax has quickly risen in popularity since it is supported by all modern browsers without requiring additional plug-ins. Like other RIAs, though, it is not well suited for search engine optimization or handling clients that have disabled client-side scripts.
- *Adobe® AIR™:* Adobe AIR (Adobe Integrated Runtime), previously code-named Adobe "Apollo," is a cross-operating system runtime that allows web



**Figure 2. JavaFX "Big Picture" (Source: Sun Microsystems, Inc.)**

application developers to use their existing web development skills with RIA technologies (such as Flash/Flex, JavaScript/Ajax, and HTML) to build and deploy RIAs to the *desktop*. While not entirely RIA, this technology is closely related to rich applications technology since it targets the same developers.

- *JavaFX*: JavaFX is a new offering from Sun Microsystems that complements the Java family of tools (see [Figure 2](#)). It addresses the growing demand in the Java community for RIA tools and technologies to deliver rich content to the client. Today the technology spans two releases: JavaFX Script and JavaFX Mobile. JavaFX Script gives Java developers the power to quickly create content-rich applications for the widest variety of clients, including mobile devices, desktops, and home electronics units. In theory, content creators now have a simple way to develop content for any Java-enabled device. JavaFX Mobile, on the other hand, is a complete software system for mobile devices.
- *Google Gears*: Google Gears is beta software offered by Google to enable offline access to services that normally work only online. It installs a database engine, based on SQLite, on the client system to locally cache web application data. Google Gears-enabled pages use data from the local cache rather than from the online service. Using Google Gears, a web application may periodically synchronize the data in the local cache with the online service whenever a network connection is available. If a network connection is unavailable, the synchronization is deferred until a network connection is established. This allows Google Gears-enabled web applications to work disconnected from the Internet, making them more like their robust desktop counterparts. While not entirely RIA, this technology is closely related to rich applications technology, as it addresses the key problem of application connectivity.

## Working with Silverlight 1.1

Now that Silverlight 1.1 is well understood and the differences between it and Silverlight 1.0 and WPF are clear, we need to dive deeper into the framework. In this section, we'll look at the inner workings of Silverlight 1.1, first reviewing the tools that have been designed to work with Silverlight, then reviewing the Silverlight design model, and finishing with a look at the Silverlight development model.

### Tools

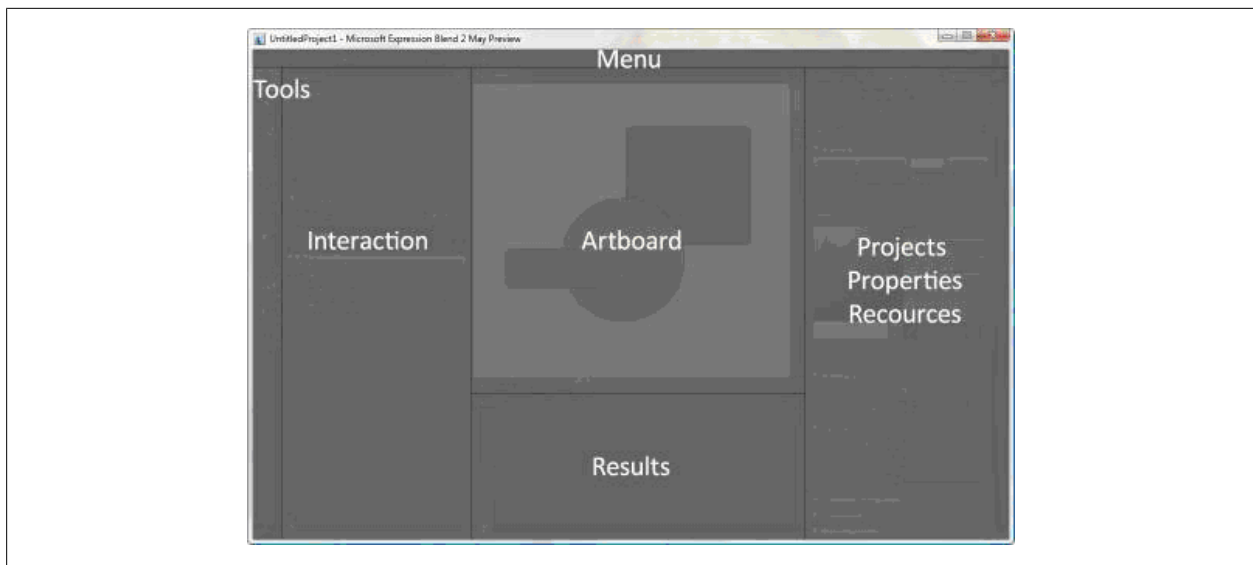
The next-generation tools being created by Microsoft have Silverlight development in mind. From the Expression products on the design side that make XAML editing a breeze to the new Visual Studio 2008 Silverlight integration, Microsoft is doing everything it can to make it easy for developers to adopt their new RIA framework. For many developers, these tools are already part of their normal development workflow, which is another key benefit Silverlight offers over traditional competitors like Flash.

#### Tools for Designers: Microsoft Expression Blend

The Expression suite of products is a relatively new offering from Microsoft, so these tools are less likely to exist in current workflows than their development counterparts. The Expression products were first introduced under codenames like "Acrylic," "Quartz," and "Sparkle" in September 2005, but since then Microsoft has made it clear that it is serious about creating tools for designers that rival long-time leader Adobe (which has acquired Macromedia since the Expression tools were originally introduced). Led by industry veteran Forest Key, who has previously worked for both Industrial Light and Magic and Adobe Systems, Microsoft definitely has the ability to deliver powerful new products that integrate seamlessly into the XAML/Silverlight workflow.

Microsoft Expression Blend, previously Expression Interactive Designer, previously codenamed Sparkle, is a collaboration tool for visually creating and editing XAML files. It is primarily used for designing interactions and animations; Expression Design is Microsoft's product designed for actually creating the visual assets that you animate in Blend. Everything created in Blend can be viewed and edited directly in Visual Studio 2008, so it is the perfect tool for enabling collaboration between developers and designers.

Like Visual Studio, Blend works with projects that can contain any number of files and visual assets. Blend is used primarily to edit *xaml* files, enabling you to use a vast array of tools to manipulate XAML assets on a canvas. Any changes you make to a project in Blend—whether it's changing an existing file or adding a new XAML file—are instantly available in Visual Studio. Developers can even open the same



**Figure 3. Microsoft Expression Blend workspace overview**

project in Visual Studio that designers use in Blend, completely eliminating the friction that usually exists between designer projects and developer projects.

## Workspace

Blend has a vector-based modular workspace consisting of several elements—Menu, Toolbox, and Artboard—and a few panels, such as Projects, Results, and Properties (see [Figure 3](#)). The presentation is a mix between Visual Studio and existing visual designer products (like Adobe Flash), so it is an easy product for developers and designers to learn.

### Workspace Areas

Let's take a closer look at the different workspace areas in Expression Blend, focusing on the tools that are available for working with Silverlight:

- *Menu area:* All Expression Blend commands and items are placed in the menu area, which still adheres to the "classic" menu model. Expression Blend has not adopted the RibbonBar approach for presenting commands and it is unclear whether the product will eventually shift to the newer UI model.
- *Toolbox area:* Blend's toolbox for Silverlight includes various instruments for manipulating objects, such as selecting, viewing, and changing appearance. Unlike other programs for graphic manipulations, in Blend tools can only change *existing* properties. The properties of the objects are initially set in the Properties panel. In other words, while in most graphic software the "paint bucket" tool is used to assign a certain fill to an object, in Blend the paint bucket copies the properties of the currently selected fill object to another object on

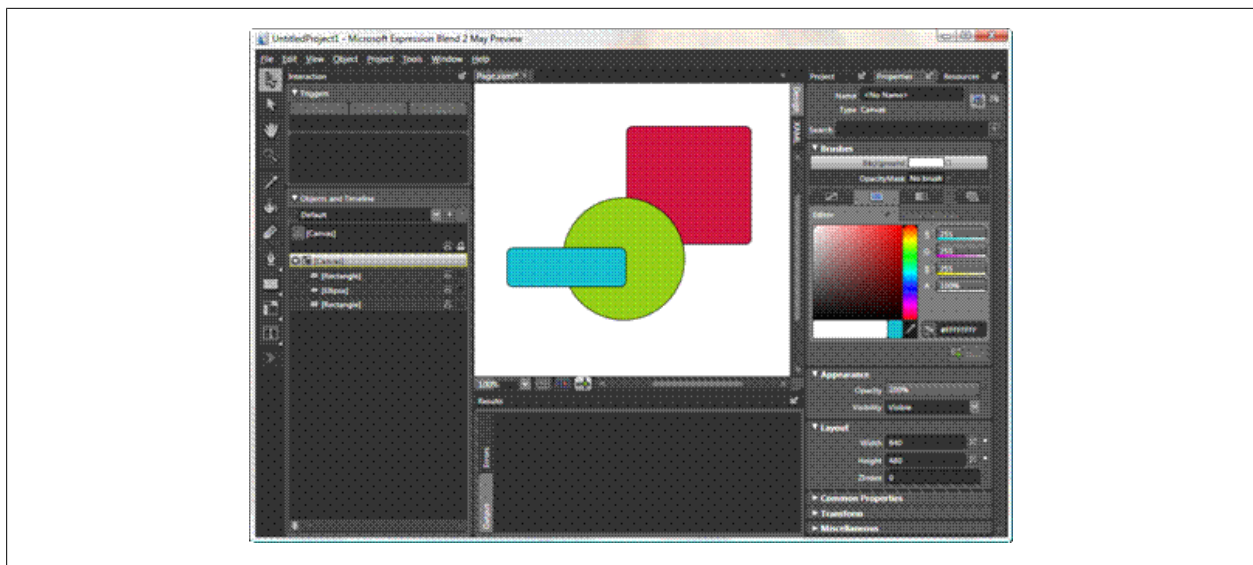


the artboard. This affects all properties—the fill, the stroke, and the opacity mask—and is a key difference between Blend and "normal" design products.

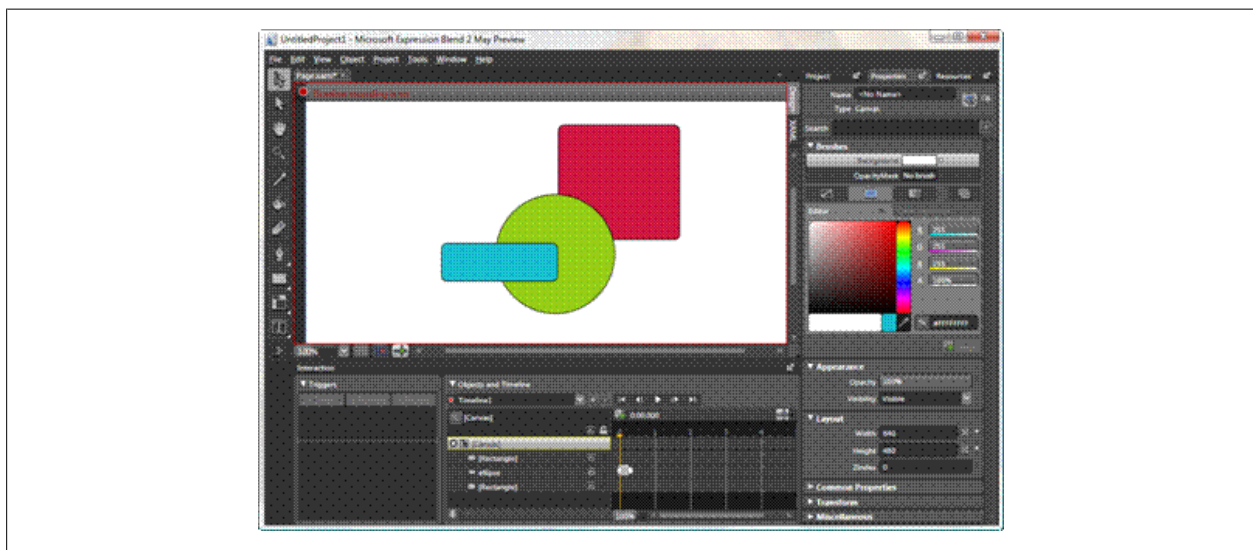
- *Interaction panel:* This panel handles all animation, interaction, and layout properties. Using these tools, you can easily add motion and interaction to your XAML. The artboard is where the XAML is visually displayed on the Canvas, so any layout changes you make or animations you apply can be seen live in that area.
- *Results panel:* To help you debug your XAML files, the results panel shows output and errors in the XAML code. This concept is much more familiar to developers than designers, so this feature is more likely to be used by developers that find themselves working in Blend.
- *Projects panel:* The Projects panel allows you to organize and manage the files in your project. This interface is virtually identical to its counterpart in Visual Studio, so it should be very familiar to developers. Also familiar to developers is the Properties panel, which gives you complete control over the visual appearance of an object.
- *Resources panel:* This panel is the place where available resources for the project are stored. Unfortunately, in the May CTP of Blend, you cannot create resources for Silverlight. This is a problem that will likely be addressed in a future update to the Blend product.

When a XAML file is opened in Blend, it is displayed in a separate window on the Blend artboard. Each window has two views: Design View and XAML View. The ability to switch between Design and XAML View helps you check the results of your code writing or see what code is generated when you create things visually. This is another very developer-oriented feature that is unique to the Blend graphics program, made possible by the open, XML-based markup that is used to define XAML objects.

Blend also has two predefined workspaces: Design Workspace (shortcut key F6) and Animation Workspace (shortcut key F7). The main difference between the two is that in the animation workspace, the Objects and Timeline panel positions horizontally to accommodate the timeline. There are a few additional shortcuts that make navigating the Blend workspace a breeze. If you need more artboard space for your project, you can quickly decrease the tools and panels size from Tools > Options > Workspace Zoom. Also, familiar to Adobe users, you can use the Tab key to toggle the visibility of all workspace panels. **Figure 4** illustrates the design workspace; **Figure 5** shows the animation workspace.



**Figure 4. Expression Blend design workspace**

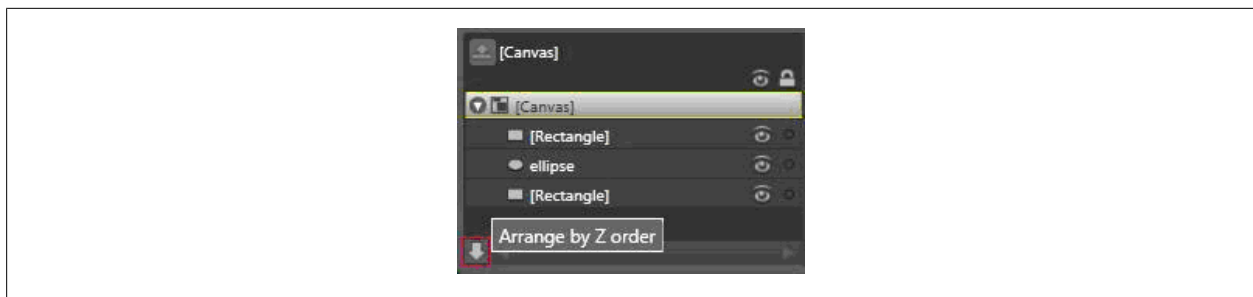


**Figure 5. Expression Blend animation workspace**

## Objects

In Silverlight, you have the ability to work with a variety of objects—vector shapes, text, raster images, audio, and video. Each object is wrapped in a Canvas element that defines its top-left position.

Objects can be edited in many ways: they can be moved, copied, pasted, deleted, aligned, animated, transformed, grouped, and ungrouped. Grouping is a very important operation in Blend since it allows you to address many objects as one. Each group has its own XAML canvas.



**Figure 6. Expression Blend Objects panel**

Objects are arranged on the artboard by markup order. When a new object is created, it appears on top of all other objects, and its markup is added to the bottom of the XAML file. You can track how objects are ordered in the Objects and Timeline panel. By default the object on top of the canvas actually appears at the bottom of the list of objects in the Objects panel. This closely tracks the XAML order, but it can be visually confusing. If you wish to arrange objects in the Objects panel by Z order, simply click the arrow pointing down in the bottom-left corner of the Objects panel (see [Figure 6](#)).

## Drawing

Though Expressions Design is Microsoft's primary tool for creating complex graphics, Blend's toolset allows you to draw most of the basic XAML shapes: line, curve, freeform path, rectangle, and ellipse. Blend has no dedicated tool for creating a polygon shape, though, so you will have to use code to draw one in Blend. For information on this topic, refer to [Silverlight Design Model](#) of this Short Cut.

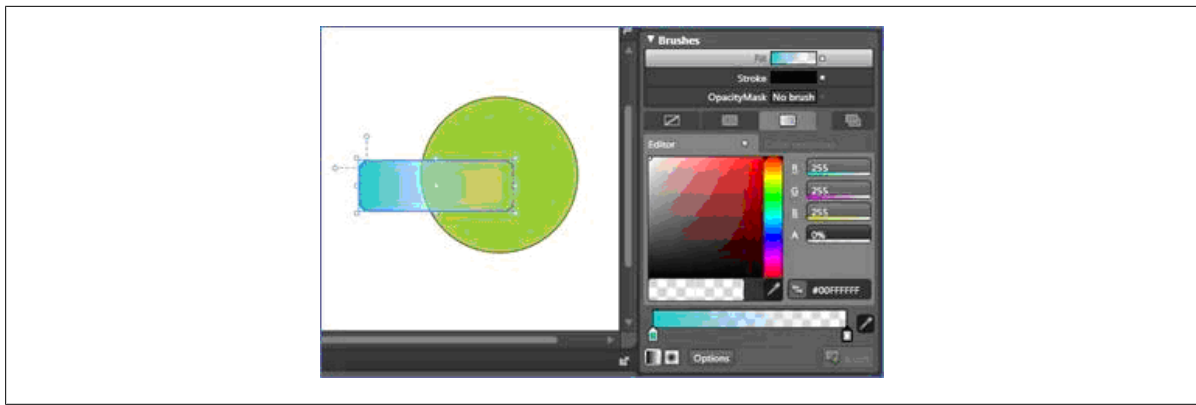
## Appearance

The visual appearance of an object in Silverlight is defined by common properties like Fill, Stroke, Opacity Mask, Foreground, Background, Opacity, and Visibility. Different objects have different properties:

- *Shapes* have Fill, Stroke, and Opacity Mask.
- *Text* objects can have Foreground and Opacity Mask, but not Stroke or Fill.
- *Images and video* can only have Opacity Mask.

## Brushes

Blend uses brushes (see [Figure 7](#)) to paint objects, a concept similar to the brushes used in GDI+ graphics creation. Brushes can be set to fill, stroke, and opacity mask properties. Blend exposes three types of brushes for Silverlight: No Brush, Solid Color Brush, and Gradient Brush. In the code behind, you can create more brushes, which we'll discuss in more detail later in this Short Cut.



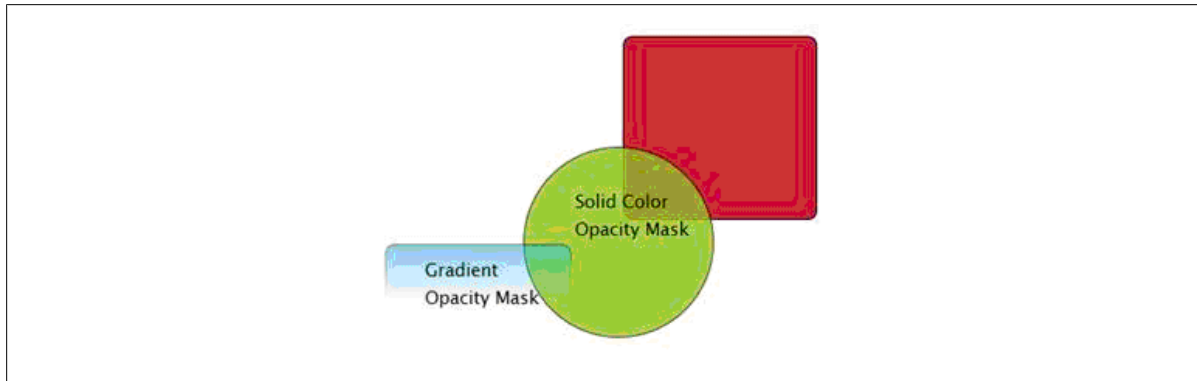
**Figure 7. XAML brushes applied to shapes in Blend**

- *No Brush* means that the selected property is not painted. It allows you to make shapes with no stroke, no fill, or no opacity mask.
- *Solid Color Brush* paints a property in a solid color.
- *Gradient Brush* paints a property in one, two, or more colors in gradation. A Gradient Brush can be linear or circular.

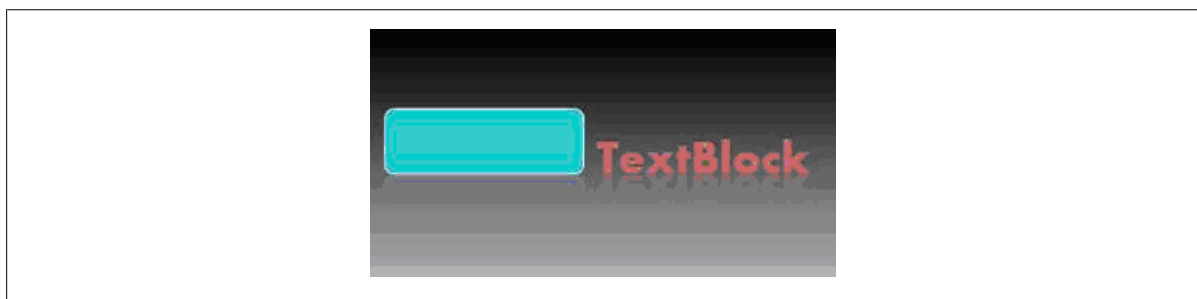
Visibility, Opacity, Transparency, and Opacity Mask are similar properties, and they all apply to an object's visible appearance. The differences are described as follows:

- **Visibility** allows showing and hiding objects completely without deleting them from the project. With the **visibility** property, objects can appear visible, hidden, or collapsed. This property is used mainly for creating interactions and animations in cases when the object should appear on or disappear from the page.
- **Opacity** affects the transparency of the *whole* object and ranges from 0 to 100 percent. This enables you to easily create semitransparent objects, which is a convenient way to make inactive button states.
- **Transparency** changes the alpha value of an object's *attributes* (such as the **Stroke** or **Fill**), unlike **opacity**, which applies changes at the object level. Also unlike **opacity**, **transparency** is not an object property. Rather, it is applied to each color, making it very useful for creating complex and realistic shading.
- **Opacity Mask** links the object visibility to the opacity mask brush (see [Figure 8](#)).

When the object is set to a solid color brush opacity mask, the result is identical to setting **opacity** directly on the object. If you use a gradient brush opacity mask, though, you can create powerful effects like reflection (see [Figure 9](#)).



**Figure 8. Opacity masks applied to shapes in Blend**




**Figure 9. Opacity masks used to create reflections in Blend**

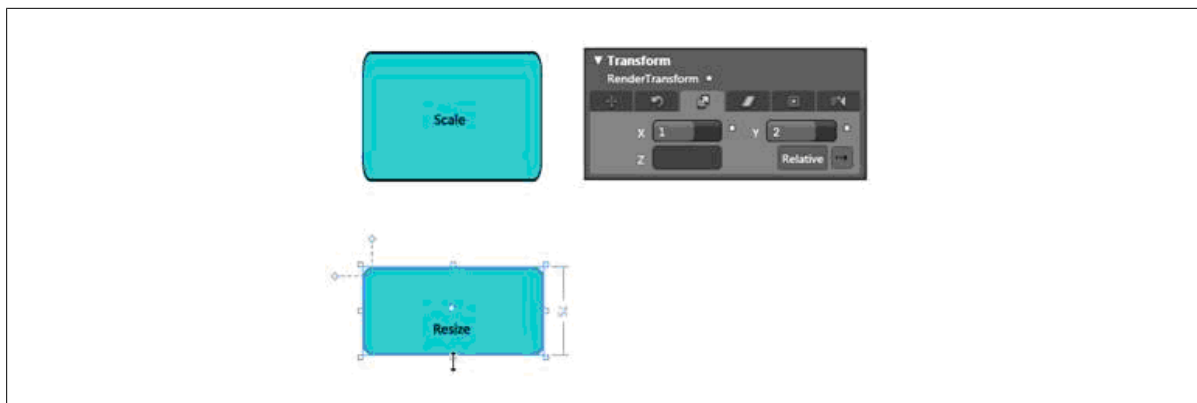


**Figure 10. Transform panel in Expression Blend**

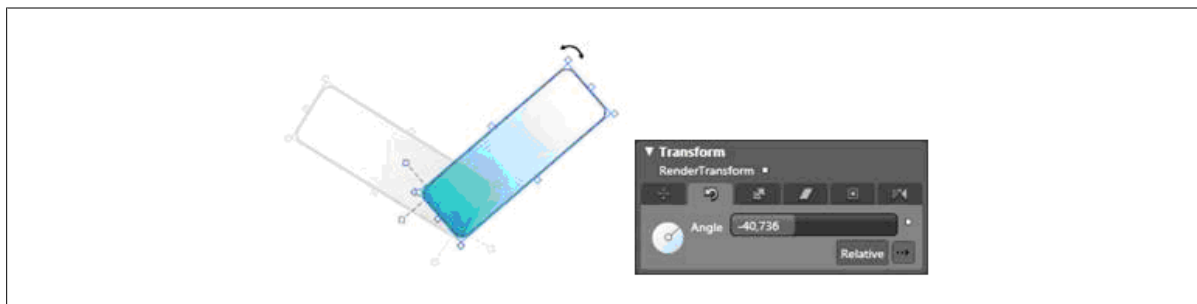
## Modifying and Transforming Objects

In Blend, you can translate, scale, rotate, skew, and flip objects by using the Selection Tool  or by using the Transform Category ([Figure 10](#)) under the Properties panel.

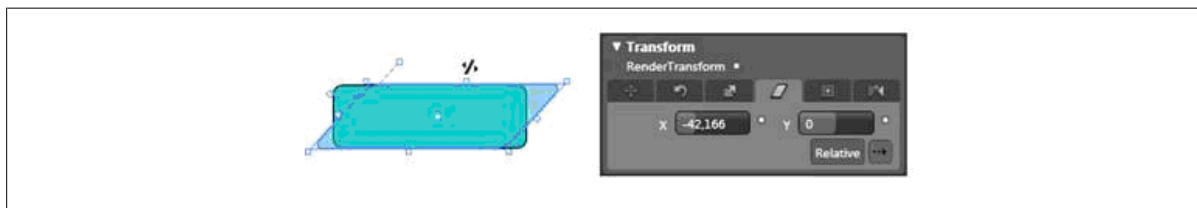
- *Translating* an object means changing the object's position along the X and Y axes. This is different from moving the object on the artboard using the selection tool, which changes the position of the object relative to its parent canvas by Top and Left values. To translate an object, you should change the X and Y values in the Transform panel.



**Figure 11. Resizing shapes versus scaling**



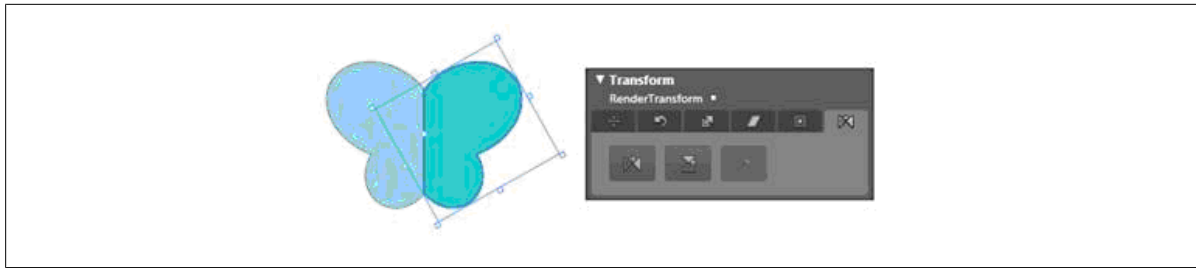
**Figure 12. Rotating an object in Expression Blend**



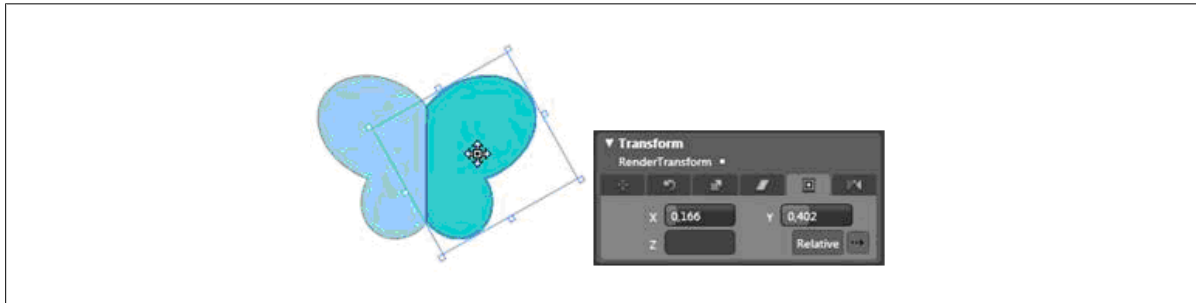
**Figure 13. Skewing an object in Expression Blend**

- *Resizing* an object means changing its **Width** and **Height** properties without applying a transformation, while scaling it transforms the object's shape. Resizing is essentially scaling with constrained proportions (see [Figure 11](#)).
- *Rotating* an object ([Figure 12](#)) can easily be done using its handles and the selection tool. When rotating visually with this tool, the object is rotated around its center point to whatever angle you desire. This is the same as changing the angle value for an object in the Transform panel.
- *Skewing* an object ([Figure 13](#)) can be done by modifying the X or Z coordinates on the Skew tab in the Transform panel or by using the Selection tool. Like rotation, either approach produces the same results.






**Figure 14. Flipping an object in Expression Blend**





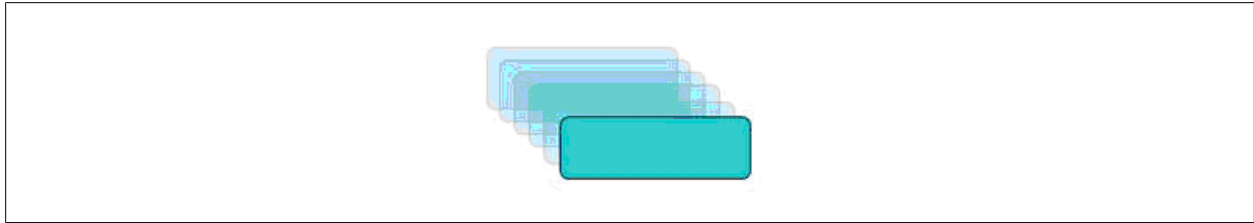
**Figure 15. Changing an object's center point in Expression Blend**

- *Flip* an object easily in Blend by using the Flip tab in the Transform panel ([Figure 14](#)). As you would expect, you can flip an object horizontally or vertically.
- *Changing the center point* of an object changes the way an object is transformed in rotation and flipping. To change the center point, simply select the object, then grab the center point and move it. For more precise results, you can use the Center Point Tab or the Center Diagram  on the Translate tab (see [Figure 15](#)).

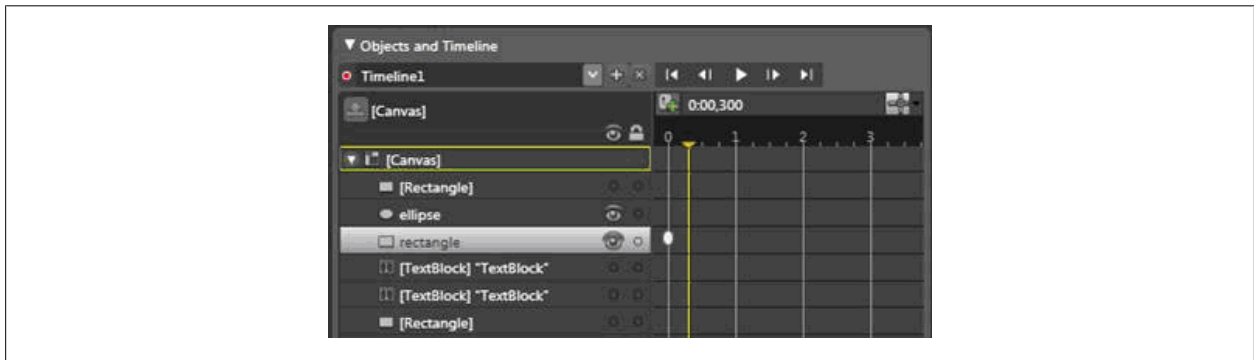
## Animation

Animations in Blend are based on keyframes recorded in a timeline. Keyframes define object property changes at specific points in time, a concept that is familiar to most motion graphics designers. The animation manifests from the interpolation of the changes occurring between the keyframes. If you want to create an animation of an object moving from point A to point B on the artboard ([Figure 16](#)), you must simply add two keyframes to the timeline—the first contains the object at point A, the second displays the object at point B. Animations are controlled from the Object and Timeline panel.

To create an animation, first create a new timeline by clicking the plus button  next to the Timeline drop-down in the Objects and Timeline Panel. Then select the object you wish to animate on the artboard and create a new keyframe by clicking the "Create New Keyframe" button .




**Figure 16. Object moving from point A to point B.**



**Figure 17. Animation timeline in Expression Blend**

Next, define a timeframe for your animation. You can do this by dragging the yellow marker in the Timeline panel or by manually setting the timer located next to the "Create New Keyframe" button.

By now you should see something that looks like [Figure 17](#).

To continue your animation, click the "Create New Keyframe" button  again. Now you have the first and the second keyframe in your animation. Transform or modify the selected object to create your animation—you can move, rotate, flip, change the color appearance, etc. Just make sure that the yellow marker in the Timeline panel is pointing to the keyframe you wish to edit. All animation changes are applied to the keyframe under the yellow marker. To preview your animation, click on the Play button on the upper portion of the Timeline panel.

### Tools for Developers: Visual Studio 2008

Unlike Blend, the Visual Studio product line is a well-established Microsoft tool. Most developers tasked with building Silverlight applications are likely already using Visual Studio in their daily workflows. The goal of this section is to demonstrate a good practice for successfully developing Silverlight applications using products that are in both alpha and beta stages, respectively. Visual Studio 2008 will ship toward the end of 2007, and Silverlight 1.1 should be in beta by early 2008. In addition to looking at Visual Studio 2008 tools, we will also briefly take a look at Visual Studio and Expression Blend integration.

Before we get started, it is important to identify which of the many preview bits are required to build Silverlight applications. Download and install these bits to prepare for successful Silverlight 1.1 development:

- *Visual Studio 2008 Beta 2* <http://msdn2.microsoft.com/en-us/vstudio/aa700831.aspx>
- *Silverlight 1.1 Alpha Refresh for Windows* This is the distributable required to run Silverlight applications. <http://www.microsoft.com/silverlight/license-win-dev.aspx>
- *Silverlight Tools Alpha for Visual Studio 2008 Beta 2* This package is an add-on to the Beta 2 release of Visual Studio 2008 that provides tools for Microsoft Silverlight 1.1 alpha. It provides a Silverlight project system for developing Silverlight applications using C# or Visual Basic. <http://www.microsoft.com/downloads/details.aspx?familyid=b52aeb39-1f10-49a6-85fc-a0a19cac99af&displaylang=en>
- *Silverlight 1.1 Alpha SDK* Contains useful documentation and examples. <http://msdn.microsoft.com/vstudio/eula.aspx?id=c8bf88e7-841c-43fd-c63d-379943617f36>

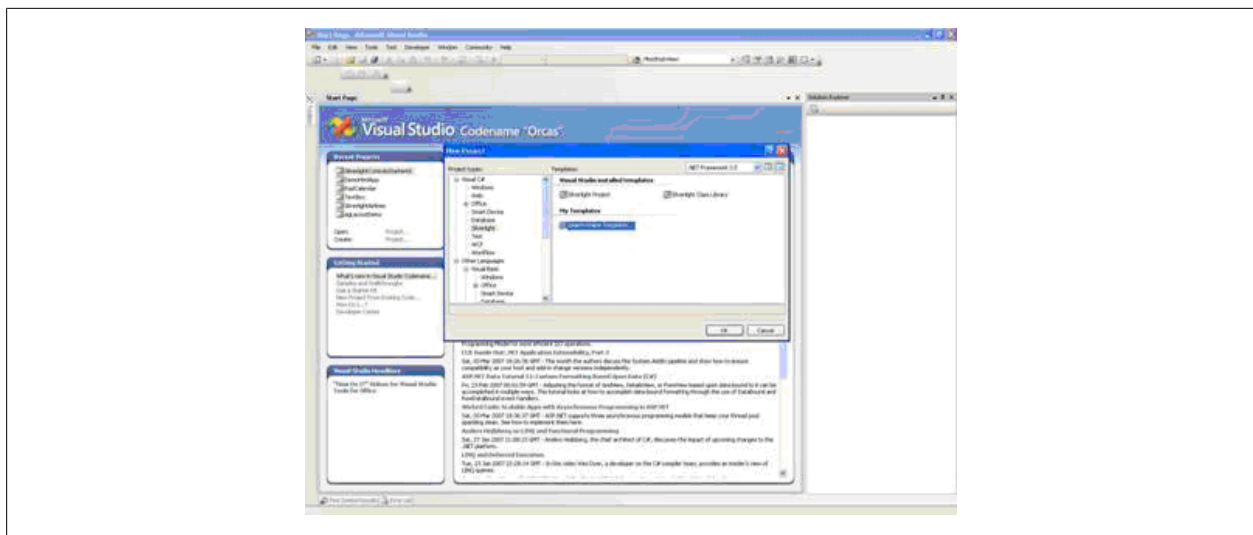
You can create Silverlight projects directly in Visual Studio 2008. Visual Studio and Expression Blend are designed to work together, and you can easily move back and forth between them, but we will discuss that process later. For now, we will focus on how to start creating a Silverlight application in Visual Studio.

To begin a new Silverlight project, open Visual Studio 2008 Beta 2 and choose File > New Project (see **Figure 18**). Select Silverlight under Project types (currently projects are available for C# and VB.NET), and select Silverlight project on the right under Templates. A new project will be created when you click the OK button.

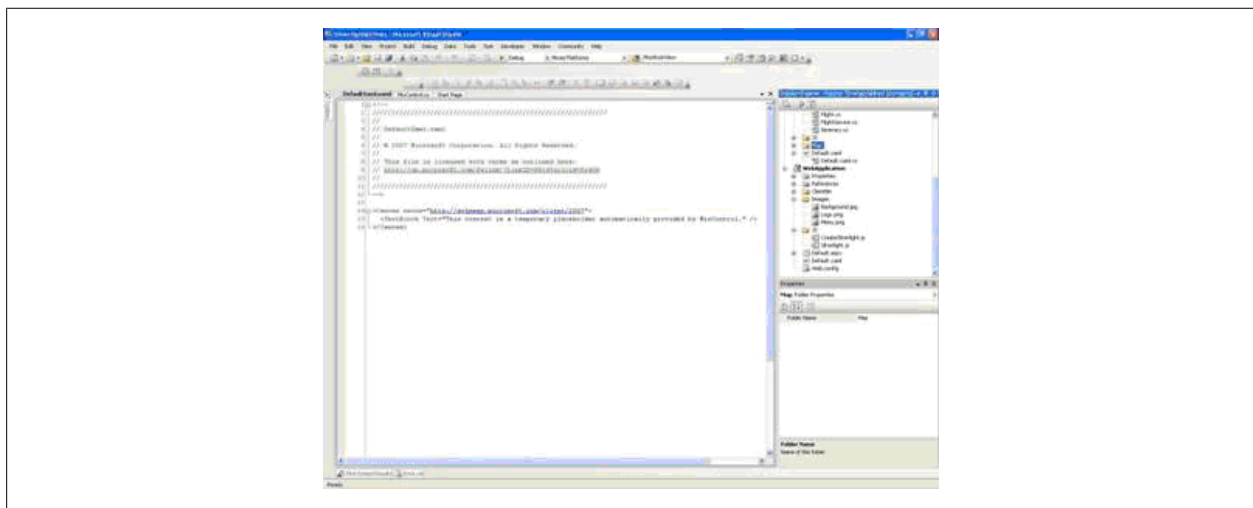
The project will be created with all of the basic items you need to begin creating a Silverlight application, such as a default XAML file and an HTML file with a Silverlight host (**Figure 19**).

## Integration of Visual Studio 2008 projects with Blend

In Visual Studio 2008 you can seamlessly switch to Blend while editing XAML files by simply selecting the file in the Solution Explorer, right-clicking, and selecting "Open in Expression Blend." Your XAML file will open in Blend where you can visually edit your XAML markup (**Figure 20**).



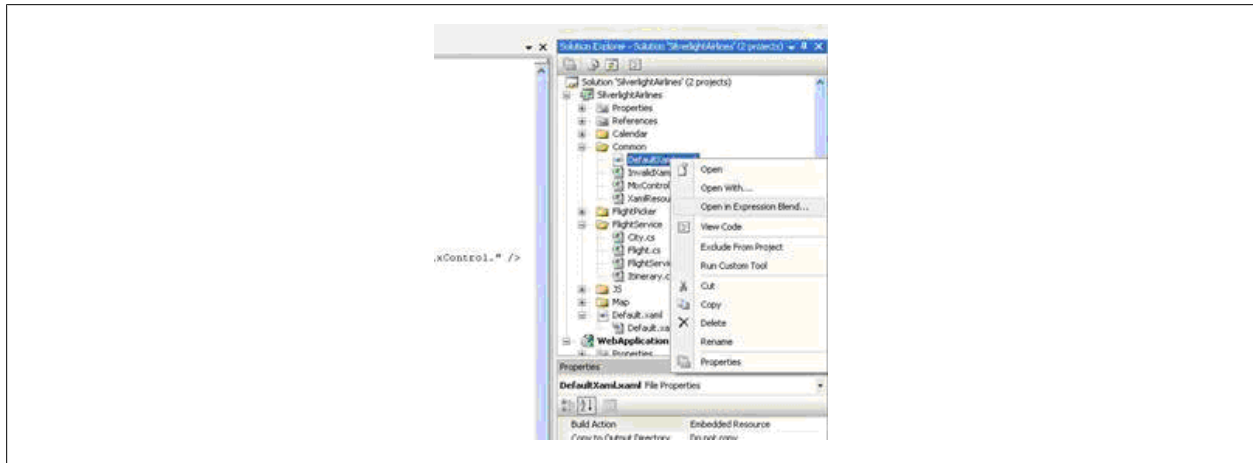
**Figure 18. Creating a new Silverlight 1.1 application in Visual Studio**



**Figure 19. A default Silverlight application in Visual Studio**

If you are working with a CTP or Beta of Visual Studio 2008, you will notice that projects and solutions that are created or edited in Visual Studio "Orcas" cannot be successfully reloaded inside Expression Blend. To overcome this issue, the Expression team at Microsoft made a utility that you can use to configure Expression Blend for Visual Studio 2008 betas. This tool can be used with the Expression Blend 2 May Preview, and it is available at <http://blogs.msdn.com/expression/attachment/2972363.ashx>. If you uninstall Visual Studio 2008 (after configuring Blend for VS2008 with this tool), the Blend product will stop working until re-configured for Visual Studio 2005 using the same tool.

We will take a more detailed look at using Visual Studio to create a Silverlight project later in this Short Cut and in the accompanying webcast. In any event, by



**Figure 20.** With a single click in the Solution Explorer, the context menu enables you to edit a XAML file directly in Blend

using Microsoft Expression Blend and Microsoft Visual Studio, designers and developers can collaborate effectively, using their unique skills to easily create Silverlight applications and controls.

## Silverlight Design Model

The core of Silverlight markup, in both 1.0 and 1.1, is XAML. XAML was introduced with WPF, and it represents a whole new way of defining graphic objects compared to the years to GDI-based graphics development. In this section, we will take a detailed look at all of the available XAML objects in Silverlight 1.1, giving you a solid foundation in the basics of XAML markup.

### Shapes

Currently Silverlight supports six vector shape elements: Rectangle, Ellipse, Line, Polygon, Path, and Polyline. Following is a detailed look at each shape, complete with sample code and images showing you how each shape is created.

- **Rectangle:** Describes a square or rectangular shape. Common properties are Width, Height, RadiusX, and RadiusY. The last two allow you to control the corner curvature of the Rectangle.



XAML

```
<Rectangle Fill="#FF144907" Stroke="#FF000000" StrokeThickness="5" RadiusX="5" RadiusY="5" />
```

- **Ellipse:** Describes an oval or circular shape. Common properties are **Width** and **Height**, which control the horizontal and vertical diameter of the **Ellipse**, respectively.



XAML  
`<Ellipse Fill="#FF144907" Stroke="#FF000000" StrokeThickness="5" Width="120" Height="120" />`

- **Line:** Common properties are **X1**, **Y1**, **X2**, and **Y2**. Point (X1, Y1) defines the *beginning* of the line, while point (X2, Y2) marks its *end*. Most elements have **fill** and **stroke** properties, but a **Line** only has the **stroke** property.



XAML  
`<Line X1="10" Y1="10" X2="150" Y2="150" Stroke="#FF000000" StrokeThickness="5" />`

- **Polygon:** Describes a collection of points. The obvious and most important property for this shape is the **Points** property. Each point is defined in comma-separated pairs (such as 5,5), separated from any other point by a space. Points are relative to the top left of the shape's parent canvas.



XAML  
`<Polygon Points="20,20 50,150 110,110 120,15 50,0" Fill="#FF144907" Stroke="#FF000000" StrokeThickness="5" />`

- **Path:** Used to describe complex shapes. The **Data** property defines the visual appearance of the element. Unlike **Polygon**'s, **Path** shapes enable more complex curved splines.





XAML

```
<Path Data="M 10,100 C 0,0 300,0 250,100z" Fill="#FF144907" Stroke="#FF000000"
StrokeThickness="5" />
```

- **Polyline:** Describes a collection of points that might or might not be connected. Like the **Polygon**, the most important property is the **Points** property. Points are defined just like **Polygons**, with comma-separated pairs separated by spaces.



XAML

```
<Polyline Points="0,0 0,100 100,100 100,0" Stroke="#FF000000" Fill="#FF144907"
StrokeThickness="5" />
```

All of these shapes could be rendered on a single canvas by using the complete XAML markup in [Example 1](#).

### Example 1. Complete XAML markup for all basic shapes

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

<Rectangle Fill="#FF144907" Stroke="#FF000000" StrokeThickness="5" RadiusX="5"
RadiusY="5" Width="180" Height="90" Canvas.Left="153.722" Canvas.Top="-16.086" />

<Ellipse Fill="#FF144907" Stroke="#FF000000" StrokeThickness="5" Width="120"
Height="120" Canvas.Left="0" Canvas.Top="0"/>

<Line X1="10" Y1="10" X2="150" Y2="150" Stroke="#FF000000" StrokeThickness="5"
Canvas.Left="216.357" Canvas.Top="-16.086"/>

<Polygon Points="20,20 50,150 110,110 120,15 50,0" Fill="#FF144907"
Stroke="#FF000000" StrokeThickness="5"
Canvas.Left="63" Canvas.Top="-16.086"/>

<Path Data="M 10,100 C 0,0 300,0 250,100z" Fill="#FF144907" Stroke="#FF000000"
StrokeThickness="5" Canvas.Left="63" Canvas.Top="35.5" Height="Auto" />

<Polyline Points="0,0 0,100 100,100 100,0" Stroke="#FF000000"
StrokeThickness="5" Width="Auto" Height="Auto" Canvas.Top="17.5"
```

```
Canvas.Left="66.857"/>
```

```
</Canvas>
```

## Brushes

Now that you understand the basic shape elements available to Silverlight, it is important to understand how to style the objects. In this section we will take a detailed look at Brushes that enable you to manipulate the **Fill**, **Stroke**, and **Opacity Mask** properties on an object, thus improving an element's visual appearance. Currently Silverlight supports five brushes: **SolidColorBrush**, **LinearGradientBrush**, **RadialGradientBrush**, **ImageBrush**, and **VideoBrush**.

- **SolidColorBrush**: Used to paint an element with a solid color. Applicable for both **Fill** and **Stroke** properties. As with many properties in XAML, there are multiple ways to achieve the same result, which is illustrated in the following code sample.



XAML

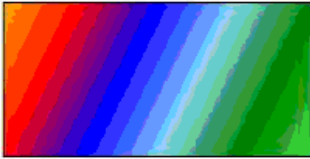
```
<Rectangle Fill="#FF490707" Stroke="#FF000000" StrokeThickness="5"  
RadiusX="10" RadiusY="10" Width="180" Height="90" />
```

Or:

```
<Rectangle Stroke="#FF000000" StrokeThickness="5" RadiusX="10"  
RadiusY="10" Width="180" Height="90" >  
  <Rectangle.Fill>  
    <SolidColorBrush Color="Red" />  
  </Rectangle.Fill>  
</Rectangle>
```

XAML supports both hexadecimal fill colors (like "#FF000000") and defined fill colors (like "Red"). Using either value is valid in XAML markup and your choice will depend on the level of precision that you need in your color selection. Among the colors that can be set by their string representations are: Black, Blue, Brown, Cyan, DarkGray, Gray, Green, LightGray, Magenta, Orange, Purple, Red, Transparent, White, and Yellow.

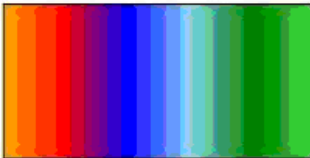
- **LinearGradientBrush**: Draws a gradient along a line. This line is horizontal by default.



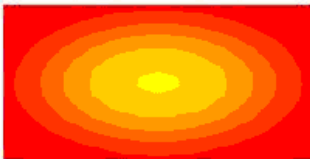
```
XAML
<Rectangle Width="200" Height="100" Stroke="Black">
  <Rectangle.Fill>
    <LinearGradientBrush>
      <GradientStop Color="Orange" Offset="0.0" />
      <GradientStop Color="Red" Offset="0.20" />
      <GradientStop Color="Blue" Offset="0.40" />
      <GradientStop Color="SkyBlue" Offset="0.60" />
      <GradientStop Color="Green" Offset="0.80" />
      <GradientStop Color="LimeGreen" Offset="1" />
    </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

Using the `StartPoint` and `EndPoint` properties, you can change the position of the line.

`<LinearGradientBrush StartPoint="0,0" EndPoint="1,0">`, will produce the following effect.



- **RadialGradientBrush:** Draws a gradient along a circle. Customizing the gradient can be done by setting the `GradientOrigin`, `Center`, `RadiusX`, and `RadiusY` properties.



```
XAML
<Rectangle Width="200" Height="100" Canvas.Left="25" Canvas.Top="200">
  <Rectangle.Fill>
    <RadialGradientBrush GradientOrigin="0.5,0.5" RadiusX="0.5"
      RadiusY="0.5">
      <GradientStop Color="Yellow" Offset="0.0" />
      <GradientStop Color="Orange" Offset="0.5" />
      <GradientStop Color="Red" Offset="1.0" />
    </RadialGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

```

        </RadialGradientBrush>
    </Rectangle.Fill>
</Rectangle>

```

- **ImageBrush:** Fills a shape with an image from a file. By default the image stretches itself in order to fill the shape. Setting the **Stretch** property enables the developer to change the stretching mode.



```

XAML
<Rectangle Width="200" Height="100" Stroke="Black" >
    <Rectangle.Fill>
        <ImageBrush ImageSource="telerikLogo.png" Stretch="Fill"/>
    </Rectangle.Fill>
</Rectangle>

```

- **VideoBrush:** The **VideoBrush** depends on the **MediaElement** for its video stream, so using a **VideoBrush** requires that you first add a **MediaElement** to your markup. The **MediaElement** should be configured with the video that you want to display and then the **VideoBrush** should be configured to display the **MediaElement** source. Furthermore, the **MediaElement** needs to have an **Opacity** set to 0 so that it does not appear on the screen when a video brush is used. The **MediaElement** is examined in more detail later in this Short Cut, but **Example 2** demonstrates how it is used with the **VideoBrush** and applied to a **Rectangle**.

### Example 2. XAML **MediaElement** and **VideoBrush** applied to a **Rectangle**

```

<MediaElement x:Name="mediaSource" Source="telerik.wmv"
    AutoPlay="true" Opacity="0" />

<Rectangle Width="200" Height="100" Stroke="Black" >
    <Rectangle.Fill>
        <VideoBrush SourceName="mediaSource" />
    </Rectangle.Fill>
</Rectangle>

```

## Transformations

Silverlight supports four transformations: **RotateTransform**, **ScaleTransform**, **SkewTransform**, and **TranslateTransform**.

- **RotateTransform:** Rotates an element by the specified angle.



XAML

```
<Rectangle RenderTransformOrigin="0.5,0.5" Fill="Green" Stroke="Black"
StrokeThickness="5" RadiusX="10" RadiusY="10" Width="160" Height="80">
    <Rectangle.RenderTransform>
        <TransformGroup>
            <RotateTransform Angle="45"/>
        </TransformGroup>
    </Rectangle.RenderTransform>
</Rectangle>
```

- **ScaleTransform:** Setting the `ScaleX` and `ScaleY` properties will resize the element by the factor you specify. Setting `ScaleX` to 2.0 stretches the element to 200 percent of its original width. A `ScaleY` value of 0.5 shrinks the height of the element by 50 percent.



*ScaleX = 1.5 ScaleY = 1.5 ScaleX = 1.0 ScaleY = 1.0*

XAML

```
<Rectangle RenderTransformOrigin="0.5,0.5" Fill="Green" Stroke="Black"
StrokeThickness="5" RadiusX="10" RadiusY="10" Width="160" Height="80">
    <Rectangle.RenderTransform>
        <TransformGroup>
            <ScaleTransform ScaleX="1.5" ScaleY="1.5"/>
        </TransformGroup>
    </Rectangle.RenderTransform>
</Rectangle>
```

- **SkewTransform:** Stretches the coordinate space in a non-uniform manner. `SkewTransform` is typically used when you want to simulate 3D in a 2D space.



XAML

```
<Rectangle RenderTransformOrigin="0.5,0.5" Fill="Green" Stroke="Black"
```

```

StrokeThickness="2.5" RadiusX="10" RadiusY="10" Width="100" Height="50">
    <Rectangle.RenderTransform>
        <TransformGroup>
            <SkewTransform AngleX="-50" AngleY="0"/>
        </TransformGroup>
    </Rectangle.RenderTransform>
</Rectangle>

```

- **TranslateTransform:** Very useful for moving objects. Use the *X* property of the *TranslateTransform* to specify the amount, in pixels, to move the element along the X-axis. Use the *Y* property to specify the amount, in pixels, to move the element along the Y-axis.



*X* = 250, *Y* = 50

```

XAML
<Rectangle RenderTransformOrigin="0.5,0.5" Fill="Green" Stroke="Black"
StrokeThickness="5" RadiusX="10" RadiusY="10" Width="160" Height="80">
    <Rectangle.RenderTransform>
        <TransformGroup>
            <TranslateTransform X="250" Y="50"/>
        </TransformGroup>
    </Rectangle.RenderTransform>
</Rectangle>

```

## Text, Image, and Media

- **TextBlock:** Text in Silverlight is usually displayed by using a lightweight control called *TextBlock*.

Hi, I am a TextBlock!

```

XAML
<TextBlock Text="Hi, I am a TextBlock!" />

```

Common *TextBlock* properties are *Width*, *Height*, *Text*, *FontFamily*, *FontSize*, *Foreground*, and *TextWrapping*.

Hi, I am a  
TextBlock!



XAML

```
<TextBlock Width="150" Height="100" TextWrapping="Wrap" FontFamily="Verdana"
FontSize="22" Text="Hi, I am a TextBlock!" />
```

Setting the TextBlock property can also be done in the following way:

XAML

```
<TextBlock>Hi, I am a TextBlock</TextBlock>
```

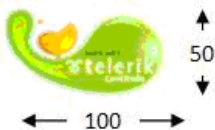
- **Image:** Images in Silverlight are displayed using the Image control. For this control to work, all you need to specify is the source image. One of the key limitations of the Image control, though, is that it supports only JPEG and PNG images. It is unclear if this limitation will be addressed in future updates to the control.



XAML

```
<Image Source="telerikLogo.png" />
```

Some of the more important properties of the Image control are Width, Height, Source, and Stretch. By default the Stretch property is set to Uniform. This means that the image will be resized according to the size of the Image control.



XAML

```
<Image Width="100" Height="50" Source="telerikLogo.png"/>
```

The Stretch property also supports None, Fill, Uniform, and UniformToFill.



XAML

```
<Image Width="100" Height="75" Source="telerik.png" Stretch="None"/>
<Image Width="100" Height="75" Source="telerik.png" Stretch="Fill"/>
<Image Width="100" Height="75" Source="telerik.png" Stretch="Uniform"/>
<Image Width="100" Height="75" Source="telerik.png" Stretch="UniformToFill"/>
```

- **MediaElement:** Displaying audio or video in Silverlight is done by using the **MediaElement** control. Like the **Image** control, all you have to set is the **Source** property of the control.



XAML

```
<MediaElement Source="telerik.wmv"/>
```

Some of the more important properties of the **MediaElement** control are **Width**, **Height**, **Source**, **Stretch**, and **Volume**. By default the **Stretch** property is set to **Uniform**, but just as with the **Image** control, this property can be changed. If a video file is used as a source, it will be automatically resized according to the size of the **MediaElement** control.

The **MediaElement** currently supports video and audio formats listed in [Table 3](#).

**Table 3. MediaElement supported file types**

Video	Audio
WMV1: Windows Media Video 7	WMA 7: Windows Media Audio 7
WMV2: Windows Media Video 8	WMA 8: Windows Media Audio 8
WMV3: Windows Media Video 9	WMA 9: Windows Media Audio 9
WMVA: Windows Media Video Advanced Profile, non-VC-1	MP3: ISO/MPEG Layer-3
WMVC1: Windows Media Video Advanced Profile, VC-1	

**MediaElements** also support playlists in the form of Advanced Stream Redirector (ASX) files. Such files typically have an *.asx*, *.wax*, *.wvx*, *.wmx*, or *.wpl* extension.

## Animations

Animation timelines are represented in XAML by a **Storyboard** element. In order to access the animation in your code behind file, you need to put the animation inside the `<Canvas.Resources></Canvas.Resources>` tag. You can access the **Story**

board in code via its name, which is specified in the `x:Name` attribute (such as `x:Name="Timeline1"`). Some of the most important attributes of an animation are `x:Name`, `BeginTime`, `Storyboard.TargetName`, `Storyboard.TargetProperty`, `KeyTime`, and `Value`. Let's look at each of these critical properties in detail:

- `BeginTime`: Shows the exact time the animation begins. It can be any valid time span value (hours/minutes/seconds).

XAML

```
BeginTime="00:02:00"
```

- `Storyboard.TargetName`: Specifies the name of the XAML element that is going to be animated. The name of the XAML element is, again, set by the `x:Name` attribute.

XAML

```
<Rectangle x:Name="rectangle1" ... ></Rectangle>
```

```
Storyboard.TargetName="rectangle1"
```

- `Storyboard.TargetProperty`: Specifies the transformation that is going to be applied. In [Example 3](#), we use the `TranslateTransform` transformation in order to move the rectangle 110 pixels (`Value="110"`) to the right (`TranslateTransform.X`) in a matter of 1 second (`KeyTime="00:00:01"`).

### Example 3. Using XAML to animate a rectangle

```
<Canvas.Resources>
  <Storyboard x:Name="Timeline1">
    <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="rectangle"
Storyboard.TargetProperty="(UIElement.RenderTransform).(TransformGroup.
Children)[0].(TranslateTransform.X)">
      <SplineDoubleKeyFrame KeyTime="00:00:00" Value="0"/>
      <SplineDoubleKeyFrame KeyTime="00:00:01" Value="110"/>
    </DoubleAnimationUsingKeyFrames>
  </Storyboard>
</Canvas.Resources>

<Rectangle x:Name="rectangle" Fill="Green" Stroke="Black" StrokeThickness="5"
RadiusX="5" RadiusY="5" Width="180" Height="90" RenderTransformOrigin="0.5,0.5" >
  <Rectangle.RenderTransform>
    <TransformGroup>
      <TranslateTransform X="0" Y="0"/>
    </TransformGroup>
  </Rectangle.RenderTransform>
</Rectangle>
```

## Silverlight 1.1 Development Model

By this point, you should have a clear understanding of the basic building blocks that are used in Silverlight programming. You've seen how Microsoft is creating new graphics tools and enhancing existing development tools to make Silverlight development easy for both designers and developers. Now we'll start putting these individual blocks together and take a broader look at the Silverlight development model.

### Hosting Silverlight in HTML

When a user visits a page that uses Silverlight for the first time, the Silverlight plug-in is installed in the user's browser (if the user allows it) and the necessary Silverlight runtime files are deployed. With the runtime installed on the client, Silverlight applications eliminate any functional dependency on a specific server-side technology (such as PHP, ASP, or JSP). That means Silverlight applications can be hosted in any page type that renders valid HTML markup to the browser.

To use the Silverlight plug-in in your web page, you will need the following files:

- *Silverlight.js*: This JavaScript file is provided and supported by Microsoft and defines the `createObject` and `createObjectEx` methods that you call to instantiate the Silverlight control in an HTML page. It also provides code to handle the client user experience if the Silverlight plug-in is not installed or if the version is older than required. You host this file on your site (or deploy it to the client), but you typically do not modify its existing methods. You could add methods and code to this file that are specific to your application, but modifications are not supported by Microsoft.
- Xaml file (page.xaml, for example): This file contains the application XAML declaration. This XAML file is referenced by the `createSilverlight` or `createSilverlightEx` method calls contained in the *Silverlight.js* file. It defines the UI that will appear as the Silverlight content. The root element tag includes at minimum xmlns definitions that are required for a working Silverlight-based application. `Canvas` is the typical root element because it provides the widest support for UI composition. If your Silverlight-based application uses managed code for event handling, the root element should include an `x:Class` attribute, which enables you to reference handlers that are defined in your managed code from XAML.
- Code behind file (*page.xaml.(cs/vb)*)  
This file contains the managed code that will be executed by the plug-in on the client. This is one of the major changes in Silverlight 1.1; Silverlight 1.0 does not support code behind execution.

To instantiate the Silverlight plug-in, a few basic steps are required, several of which are new to the Silverlight 1.1 alpha refresh:

1. Reference the *Silverlight.js* file in the head section of the html page:

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

2. Place the following HTML code where you want the Silverlight plug-in to be displayed :

```
HTML/JavaScript
<div id="silverlightcontrolhost" >
  <script type="text/javascript">
    Silverlight.createObjectEx({
      source: "Page.xaml",
      parentElement: document.getElementById("SilverlightControlHost"),
      id: "SilverlightControl",
      properties: {
        width: "100%",
        height: "100%",
        version: "1.1",
        enableHtmlAccess: "true"
      },
      events: {
      }
    });
  </script>
</div>
```

The available properties are:

- **width:** The width of the Silverlight plug-in
- **height:** The height of the Silverlight plug-in
- **version:** The required plug-in version that should run the application
- **background:** The background color of the plug-in
- **isWindowless:** Whether the plugin is displayed as window-less
- **framerate:** Maximum number of frames to render per second
- **ignoreBrowserVer:** Enables/disables checking for supported Silverlight browsers and browser versions
- **inplaceInstallPrompt:** Whether the in-place install prompt appears if the specified version of the Silverlight control is not installed

- **enableHtmlAccess:** Whether the hosted content in the Silverlight control has access to the browser Document Object Model (DOM)

The available events are:

- **onLoad:** Specifies the event handling function for the control's **OnLoad** event
  - **onError:** Specifies a user-defined JavaScript error handling function that is invoked when an error is generated in the Silverlight runtime components
3. Add JavaScript to your page that provides focus to the Silverlight control when the page loads. The approach for providing focus to the Silverlight control has changed in the Silverlight 1.1 alpha.

*HTML/JavaScript*

```
document.body.onload = function() {
    var silverlightControl =
        document.getElementById('SilverlightControl');
    if (silverlightControl)
        silverlightControl.focus();
}
```

### The Basics of a Silverlight 1.1 Project

When creating a Silverlight 1.1 project in Visual Studio 2008, you should choose to create a new "Silverlight Project" from the available project templates. When the project is created, several files are added by default that are basic to a Silverlight application:

- *TestPage.html:* This is a sample HTML page that hosts the Silverlight XAML. This file can be replaced by your own Silverlight hosting pages, so it is not required for a Silverlight project to function.
- *TestPage.html.js:* This is a JavaScript file with specific code needed to create a Silverlight object from the *Page.xaml* file. This JavaScript file uses a function that is declared in the *Silverlight.js* file to create the new Silverlight instance on the page at runtime. This JavaScript could also be included directly in the HTML file, so using this approach is a matter of choice.
- *Silverlight.js:* This is the JavaScript file with general code for creating Silverlight objects. The same JavaScript file is distributed with every Silverlight application, and it is required for a Silverlight application to work correctly.
- *Page.xaml:* This is the XAML file that is hosted by default in the *TestPage.html* file. This file contains all Silverlight XAML markup that has been created in Expression Blend or Visual Studio.



- *Page.xaml.cs/Page.xaml.vb*: This is the "code behind" of the *Page.xaml* file. This file compiles into an assembly that is referenced by default in the *Page.xaml* file. The class defined in this file is that specified in the *x:Class* value in *Page.xaml*. For example, in the following code snippet, the class "Page" defined in the code behind is referenced in the *x:Class* attribute in the XAML file.

```
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="MyProject.Page;assembly=ClientBin/MyProject.dll"
...
```

*Page.xaml.cs/.vb*

```
C#
namespace MyProject
{
    public partial class Page : Canvas
    {
        public void Page_Loaded(object o, EventArgs e)
        {
            // Required to initialize variables
            InitializeComponent();
        }
    }
}

VB
Namespace MyProject
    Public Partial Class Page
        Inherits Canvas
        Public Sub Page_Loaded(ByVal o As Object, ByVal e As EventArgs)
            'Required to initialize variables
            InitializeComponent();
        End Sub
    End Class
End Namespace
```

When the application is loaded, the first method that will be executed is the *Page\_Loaded* method. You should place any initialization instructions in this method. Code behind files for XAML that execute managed code on the client are new to Silverlight 1.1, though not required. Silverlight 1.0 does not support this type of code behind approach.

When you deploy your Silverlight application, you should upload all the files except the code behind files to your web server. Your code behind code will be

compiled into a single assembly that is usually located in the *ClientBin* directory. Upload this instead of the code behind files when you deploy your application.

## Silverlight Objects

In this section, we'll look at how you interact with the Silverlight objects that are loaded on the page.

### Accessing elements and setting properties

In order to access an element from the XAML in your code behind file, you need to set the `x:Name` property of the element.

*XAML*

```
<Rectangle x:Name="rectangle" Fill="Green" Stroke="Black" Width="100" Height="25" />
```

This element is fully accessible in your code behind file and you can set all of its properties programmatically. For example, you can attach a `MouseLeftButtonUp` handler and, when the mouse is clicked, change the `Fill` of the rectangle. To do that, start by registering the event in the `Page_Loaded` method like this:

*C#*

```
this.rectangle.MouseLeftButtonUp += new MouseEventHandler(ChangeFill);
```

*VB*

```
AddHandler Me.rectangle.MouseLeftButtonUp, AddressOf ChangeFill
```

Then create a method to handle the `MouseLeftButtonUp` event, like this:

*C#*

```
private void ChangeFill(object sender, MouseEventArgs e)
{
    this.rectangle.Fill = new SolidColorBrush(Colors.Blue);
}
```

*VB*

```
Private Sub ChangeFill(ByVal sender As Object, ByVal e As MouseEventArgs)
    Me.rectangle.Fill = New SolidColorBrush(Colors.Blue)
End Sub
```



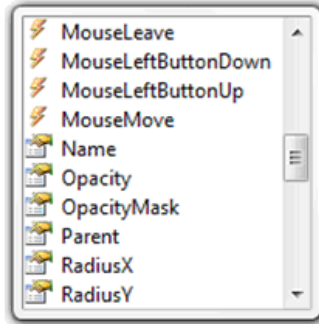
Before click    After click

You do not have to create event handlers programmatically. You can also declaratively subscribe to event handlers in the XAML markup, like this:

*XAML*

```
<Rectangle x:Name="rectangle" Fill="Green" Stroke="Black" Width="100"
Height="25" MouseLeftButtonUp="ChangeFill" />
```

```
//handle mouse click
private void ChangeFill(object sender, MouseEventArgs e)
{
    this.rectangle.Fill = new SolidColorBrush(Colors.Blue);
    this.rectangle.Width = 300;
    this.rectangle.Height = 100;
    this.rectangle.
```



**Figure 21. XAML object IntelliSense in Visual Studio 2008**

As with the programmatic approach, you have to create a `ChangeFill` method in the code behind file to handle the `MouseLeftButtonUp` event. If you are unsure which methods or properties are available for a specific object, Visual Studio's IntelliSense ([Figure 21](#)) will help you quickly determine which ones are supported.

## Create new element programmatically

Everything that can be done in the XAML file can also be done in the code behind. What makes XAML powerful is that the markup model is the same as the object model. In fact, XAML is unique from many markup languages in that it directly represents the instantiation of managed objects.

*XAML*

```
<TextBlock FontSize="20" Text="Hi, I am a TextBlock!" />
```

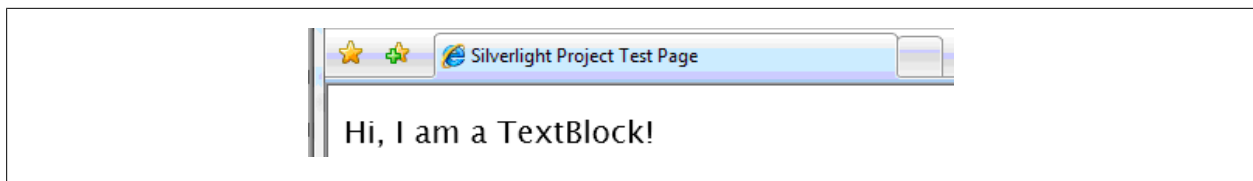
*C#*

```
TextBlock txtBlock = new TextBlock();
txtBlock.FontSize = 20;
txtBlock.Text = "Hi, I am a TextBlock!";
```

*VB*

```
Dim txtBlock As New TextBlock()
txtBlock.FontSize = 20
txtBlock.Text = "Hi, I am a TextBlock!"
```

One extra line is necessary in the programmatic approach to render the `TextBlock` on the page, though. You need to add the newly created `TextBlock` to the `Children` collection of the `Canvas` it is in—in our case, the root `Canvas`. So immediately after setting the `TextBlock`'s `Text` property, add:



**Figure 22. XAML TextBlock rendered by Silverlight in IE7**

```
C#
this.Children.Add(txtBlock);
```

```
VB
Me.Children.Add(txtBlock)
```

Putting it all together, the whole code block necessary to create a TextBlock and render it when the page loads can be seen in [Example 4](#).

#### **Example 4. Programmatically adding XAML TextBlock to a Canvas**

```
C#
public void Page_Loaded(object o, EventArgs e)
{
    // Required to initialize variables
    InitializeComponent();
    TextBlock txtBlock = new TextBlock();
    txtBlock.FontSize = 20;
    txtBlock.Text = "Hi, I am a TextBlock!";
    this.Children.Add(txtBlock);
}
```

```
VB
Public Sub Page_Loaded(ByVal o As Object, ByVal e As EventArgs)
    ' Required to initialize variables
    InitializeComponent()
    Dim txtBlock As New TextBlock()
    txtBlock.FontSize = 20
    txtBlock.Text = "Hi, I am a TextBlock!"
    Me.Children.Add(txtBlock)
End Sub
```

When you view the test page in the browser, you will see something similar to [Figure 22](#).

### **Controlling animations and MediaElements**

Displaying and controlling video in a Silverlight application is easy with the `MediaElement` control. The control exposes a number of methods for controlling video actions like play, pause, and stop, but by default it does not expose any interface for users to invoke these actions. In this example we will look at the code that is required to play/pause a `MediaElement` when it is clicked.

1. To begin, create a `MediaElement`, set its `Source` property to a video file, and attach a `MouseLeftButtonUp` event handler. The `MediaElement` can be created declaratively in the XAML code or programmatically in the code behind file.

```
XAML
<MediaElement x:Name="mediaElement" Source="telerik.wmv" AutoPlay="True"
MouseLeftButtonUp="PlayPauseVideo" />
```

Or:

```
C#
public void Page_Loaded(object o, EventArgs e)
{
    // Required to initialize variables
    InitializeComponent();
    MediaElement mediaElement = new MediaElement();
    mediaElement.SetValue<string>(MediaElement.SourceProperty, "telerik.wmv");
    mediaElement.AutoPlay = true;
    this.Children.Add(mediaElement);

    mediaElement.MouseLeftButtonUp += new MouseEventHandler(PlayPauseVideo);
}

VB
Public Sub Page_Loaded(ByVal o As Object, ByVal e As EventArgs)
    ' Required to initialize variables
    InitializeComponent()
    Dim mediaElement As New MediaElement()
    mediaElement.SetValue(Of String)(MediaElement.SourceProperty, "telerik.wmv")
    mediaElement.AutoPlay = True
    Me.Children.Add(mediaElement)

    AddHandler mediaElement.MouseLeftButtonUp, AddressOf PlayPauseVideo
End Sub
```

2. The only other step required is to create a method to handle the `MouseLeftButtonUp` event. This event will handle the mouse click, and pause or play the video depending on its current state.

```
C#
private void PlayPauseVideo(object sender, MouseEventArgs e)
{
    switch (mediaElement.CurrentState)
    {
        case "Playing":
            mediaElement.Pause();
            break;
        case "Paused":
            mediaElement.Play();
            break;
    }
}
```

```

}

VB
Private Sub PlayPauseVideo(ByVal sender As Object, ByVal e As MouseEventArgs)
    Select Case mediaElement.CurrentState
        Case "Playing"
            mediaElement.Pause()
        Case "Paused"
            mediaElement.Play()
    End Select
End Sub

```

The concept behind controlling animations is similar. In the following example, the animation starts as soon as the page loads. It displays an ellipse that moves from one position to another along the X-axis.

1. Begin this test by creating an ellipse to animate. Since the ellipse's movement will be animated, a `Translate` transformation will be used to create the motion.

```

XAML
<Ellipse x:Name="ellipse" RenderTransformOrigin="0.5,0.5" Fill="Green"
Stroke="Black" Width="60" Height="60" Canvas.Left="0" Canvas.Top="0"
MouseLeftButtonUp="PlayAnimation">
    <Ellipse.RenderTransform>
        <TransformGroup>
            <TranslateTransform X="0" Y="0"/>
        </TransformGroup>
    </Ellipse.RenderTransform>
</Ellipse>

```

2. Next, create the animation and set `Storyboard.TargetName="ellipse"`. This will instruct the animation actions to target the ellipse control. Place the animation inside `Canvas.Resources` so that it can be controlled from code.

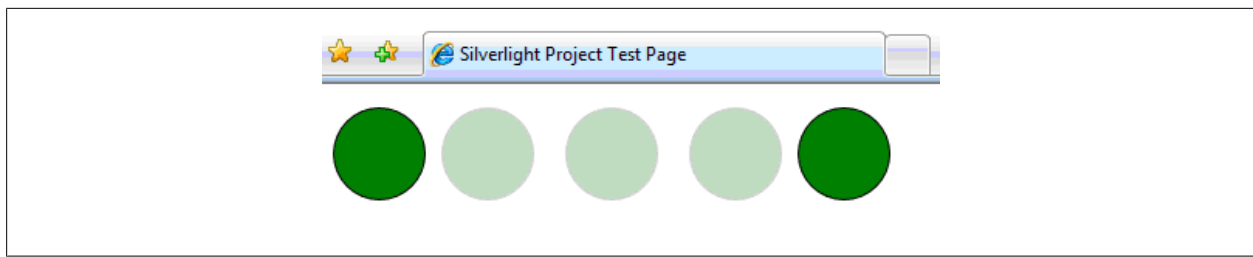
```

XAML
<Canvas.Resources>
    <Storyboard x:Name="EllipseAnimation">
        <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
            Storyboard.TargetName="ellipse"
            Storyboard.TargetProperty="(UIElement.RenderTransform).
            (TransformGroup.Children)[0].(TranslateTransform.X)">
            <SplineDoubleKeyFrame KeyTime="00:00:01" Value="300"/>
        </DoubleAnimationUsingKeyFrames>
    </Storyboard>
</Canvas.Resources>

```

3. Finally, create a method to play the animation when the ellipse is clicked. We specified the name of this event in the `MouseLeftButtonUp` property in the XAML markup.





**Figure 23. Ellipse Translate animation in Silverlight**

```
C#
private void PlayAnimation(object sender, MouseEventArgs e)
{
    this.EllipseAnimation.Begin();
}

VB
Private Sub PlayAnimation(ByVal sender As Object, ByVal e As MouseEventArgs)
    Me.EllipseAnimation.Begin()
End Sub
```

Now when you click on the ellipse, it will travel 300 pixels along the x-axis (see [Figure 23](#)).

## Layouts

The current alpha version of Silverlight 1.1 supports only the `Canvas` layout, but according to publicly available information, the official release of Silverlight 1.1 will support additional layout types. If WPF is any indication, we may also see `Grid`, `StackPanel`, and `ViewBox` layouts in Silverlight when 1.1 finally ships. More information on these layout types can be found by looking at the existing equivalents in WPF.

The sole purpose of the `Canvas` layout is to contain and position other objects. When an object is placed in a `Canvas` layout, its `Top` and `Left` properties are evaluated relative to the `Canvas` and then it is positioned in the correct place. In the following example, the red rectangle will appear at point [left: 100px, top: 80px]. The distance between the left edge of the rectangle and its parent `Canvas` will be 100 pixels, and the distance between the top edge of the rectangle and its parent `Canvas` will be 80 pixels.

```
XAML
<Canvas>
    <Rectangle Fill="Red" Width="100" Height="50" Canvas.Left="100"
        Canvas.Top="80"></Rectangle>
</Canvas>
```

The `Left` and `Top` properties are "attached" properties, and that's why you need to specify their full names when you set them (`Canvas.Top` and `Canvas.Left`). Attached

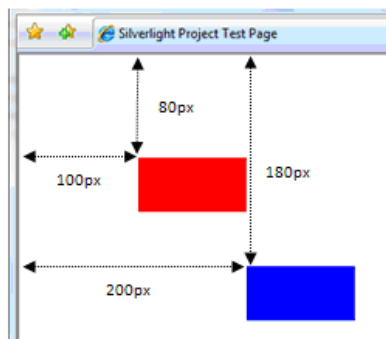
properties were developed to support the flexible layout system in WPF, and they enable you to set properties relative to an object's parent, even if the properties don't directly exist in a control's class.

In addition to placing controls in a `Canvas`, you can also place one `Canvas` into another. Each child `Canvas` will be offset from its parent `Canvas` by pixels set in its `Top` and `Left` properties. Any controls contained in the child `Canvas` will remain relative to their immediate parent. In the following example, the red rectangle will appear at point [left: 100px, top: 80px] and the blue rectangle will appear at point [left: 200px, top: 180px]. The position of the blue rectangle is calculated by summing the position of inner `Canvas` with the position of the blue rectangle.

XAML

```
<Canvas>
  <Rectangle Fill="Red" Width="100" Height="50" Canvas.Left="100"
    Canvas.Top="80"></Rectangle>

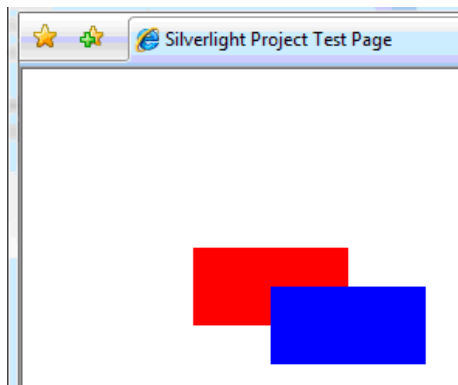
  <Canvas x:Name="innerCanvas" Canvas.Left="100" Canvas.Top="100">
    <Rectangle Fill="Blue" Width="100" Height="50" Canvas.Left="100"
      Canvas.Top="80"></Rectangle>
  </Canvas>
</Canvas>
```



The arrangement of objects on a `Canvas` is determined by the order of the XAML markup. Objects that are added later in the XAML sequence will be rendered above other objects. If the objects overlap, the objects latest in the XAML sequence will overlap other objects, which can be confusing since the order in the XAML is counter to the display order. In the following example, the blue rectangle will appear in front of the red rectangle since it is declared later.

XAML

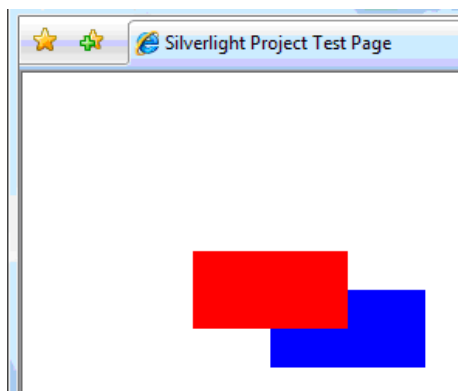
```
<Canvas>
  <Rectangle Fill="Red" Width="100" Height="50" Canvas.Left="100"
    Canvas.Top="100"></Rectangle>
  <Rectangle Fill="Blue" Width="100" Height="50" Canvas.Left="150"
    Canvas.Top="125"></Rectangle>
</Canvas>
```



The default order can be changed, though, by specifying the `Canvas.ZIndex` property. Objects with a higher `ZIndex` property will appear in front of objects with a lower `ZIndex` property. In the same example, if we set the `Canvas.ZIndex` property of the red rectangle to 1, it will appear in front of the blue rectangle because the blue rectangle will have the default `ZIndex` value of 0.

*XAML*

```
<Canvas>
<Rectangle Fill="Red" Width="100" Height="50" Canvas.Left="100" Canvas.Top="100"
Canvas.ZIndex="1"></Rectangle>
<Rectangle Fill="Blue" Width="100" Height="50" Canvas.Left="150"
Canvas.Top="125"></Rectangle>
</Canvas>
```



## Event Model

As we've seen in other sections in this Short Cut, Silverlight enables you to attach event handlers to object events programmatically or declaratively. If you opt for programmatic event attaching, the best place to attach an event handler to an existing object is in the `Page_Loaded` method. For example, if you have the following rectangle declared in your XAML:

*XAML*

```
<Rectangle x:Name="redRectangle" Fill="Red"
```

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

You can attach an event handler to the `MouseLeftButtonDown` event of the rectangle in the `Page_Loaded` method programmatically with the following code:

```
C#
public void Page_Loaded(object o, EventArgs e)
{
    // Required to initialize variables
    InitializeComponent();
    this.redRectangle.MouseLeftButtonDown += new
        MouseEventHandler(redRectangle_MouseLeftButtonDown);
}

void redRectangle_MouseLeftButtonDown(object sender, MouseEventArgs e)
{
    // Here you could place your custom code which will be executed
    // when the user clicks over the rectangle.
}

VB
Public Sub Page_Loaded(ByVal o As Object, ByVal e As EventArgs)
    ' Required to initialize variables
    InitializeComponent()
    AddHandler Me.redRectangle.MouseLeftButtonDown, AddressOf
        redRectangle_MouseLeftButtonDown
End Sub

Sub redRectangle_MouseLeftButtonDown(ByVal sender As Object,
ByVal e As MouseEventArgs)
    ' Here you could place your custom code which will be executed
    ' when the user clicks over the rectangle.
End Sub
```

You can also attach event handlers declaratively in the XAML, like this:

```
XAML
<Rectangle x:Name="redRectangle" Fill="Red" Width="100" Height="50"
Canvas.Left="100" Canvas.Top="100"
MouseLeftButtonDown="redRectangle_MouseLeftButtonDown"></Rectangle>
```

If you attach an event declaratively, you do not need to add any code in the `Page_Loaded` method. The event will automatically be wired up to the method name that you supply, requiring only that you create the method to handle the event.

Depending on the Silverlight objects that you use, different events will be available for you to handle. For example the `Canvas`, `TextBlock`, and `Rectangle` objects provide the following events:

- **Loaded:** Fires when the object is initially loaded

- **MouseEnter**: Fires when the mouse is over the object
- **MouseLeave**: Fires when the mouse is no longer over the object
- **MouseLeftButtonDown**: Fires when an object is clicked
- **MouseLeftButtonUp**: Fires when an object click is released
- **MouseMove**: Fires when the mouse is moving over the object

Meanwhile, the **MediaElement** object provides these events:

- **BufferingProgressChanged**: Fires when the buffering progress of the **MediaElement** changes
- **CurrentStateChanged**: Fires when the state of the **MediaElement** changes
- **MarkerReached**: Fires when a media marker is reached
- **MediaEnded**: Fires when the media has finished playing
- **MediaFailed**: Fires when there is an error playing the media
- **MediaOpened**: Fires when the media is initially opened

Other events like **KeyDown**, **KeyUp**, **GotFocus**, and **LostFocus** can only be defined for the root **Canvas** and will not be fired by inner **Canvases**. That means you must define a **KeyDown** event on the root **Canvas** if you want to collect keyboard input from users of a Silverlight application. Events can be attached to the root **Canvas** in the XAML markup like this:

*XAML*

```
<Canvas x:Name="parentCanvas"
xmlns=http://schemas.microsoft.com/client/2007
xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
Loaded="Page_Loaded"
KeyDown="Page_KeyDown"
...

```

*C#*

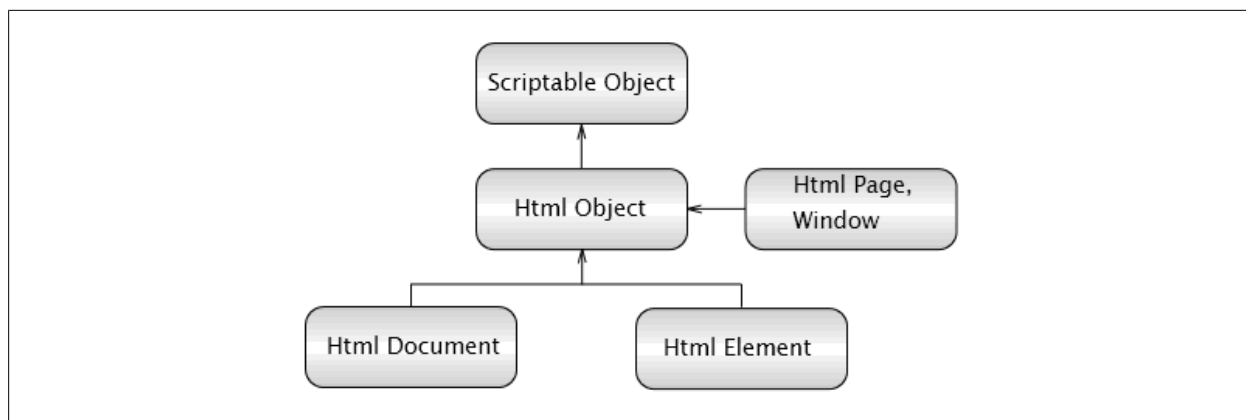
```
void Page_KeyDown(object sender, KeyboardEventArgs e)
{
    // Here you could place your custom code which will be executed
    // when the user presses any key.
}

```

*VB*

```
Sub Page_KeyDown(ByVal sender As Object, ByVal e As KeyboardEventArgs)
    ' Here you could place your custom code which will be executed
    ' when the user presses any key.
End Sub

```



**Figure 24. New .NET classes that enable HTML and Silverlight interaction**

It is important to note that once a Silverlight control is displayed in full-screen mode, keyboard events are prevented from being passed to keyboard event handlers in an application. The only valid keyboard input in full-screen mode is the set of keystrokes that returns the Silverlight control to embedded mode (such as the Escape key). This input limitation during full-screen mode is a security feature, and is intended to minimize the possibility of users entering information unintentionally into a malicious site.

### HTML Integration (DOM, Silverlight, JavaScript)

Through the use of client-side scripting, you can access and manipulate the managed code object defined by the Silverlight runtime. You can also access and manipulate the browser DOM from the managed code. **Figure 24** demonstrates the new .NET classes that enable HTML and Silverlight to interact.

All of the magic necessary to accomplish this is contained in a new .NET namespace introduced with Silverlight 1.1, called `System.Windows.Browser`. Here you'll find a number of classes that enable you to manipulate the DOM—in particular, `HtmlPage` (representing the parent browser), `HtmlDocument` (the root element of the DOM), and `HtmlElement` (for manipulating the individual elements within a page). With these classes, it is possible to create an HTML Silverlight page where all of the underlying code logic is written in a managed language (like C# or VB.NET) without writing so much as an event handler in JavaScript.

### Accessing DOM from managed code

To more clearly understand how you can access HTML elements on a page from Silverlight 1.1 managed code, let's look at some code examples. In this example, we will start with an HTML page that contains a button named "myButton". From the managed code you can access the button in the following way:

```
C#
HtmlDocument hdoc = HtmlPage.Document;
```



```
HtmlElement myButton = hdoc.GetElementById("myButton");
```

*VB*

```
Dim hdoc As HtmlDocument = HtmlPage.Document
Dim myButton As HtmlElement = hdoc.GetElementById("myButton")
```

With a reference to the button in the code behind, we can now attach an event to the button using the following code:

*C#*

```
myButton.AttachEvent("onclick", new EventHandler<HtmlEventArgs>
(this.OnMyButtonClick));
```

```
public void OnMyButtonClick (object sender, HtmlEventArgs args)
{
    // handle the button click
}
```

*VB*

```
myButton.AttachEvent("onclick", New EventHandler
(Of HtmlEventArgs)(Me.OnMyButtonClick))
```

```
Public Sub OnMyButtonClick(ByVal sender As Object, ByVal args As HtmlEventArgs)
    ' handle the button click
End Sub
```

With the same reference, we can also get and set property values on the HTML button, like this:

*C#*

```
myButton.SetProperty("value", "My Value");
string myButtonValue = myButton.GetProperty<string>("value");
myButton.SetStyleProperty("color", "#000066");
```

*VB*

```
myButton.SetProperty("value", "My Value")
Dim myButtonValue As String = myButton.GetProperty(Of String)("value")
myButton.SetStyleProperty(color, "#000066")
```

Finally, we can even manipulate the HTML page, redirecting users to a new page with this simple Silverlight code:

*C#*

```
HtmlPage.Navigate("http://www.silverlight.net");
```

*VB*

```
HtmlPage.Navigate("http://www.silverlight.net")
```

## Accessing managed code from DOM

On the other side of the coin is accessing managed code from the DOM. This ability gives client-side scripts access to the extensive library of .NET Framework features

that are available in managed code, all without requiring a round trip to the web server.

To enable this functionality, the following steps are required:

- Add a reference to the `System.Windows.Browser` namespace to the XAML code behind file.
- Add the `[Scriptable]` attribute to the class and the event, method, or property you want to expose.
- Register the scriptable object using `WebApplication.Current.RegisterScriptableObject`.

In **Example 5**, we will create a method called "Test" in our `Page` class and expose it to the client-script interface. We will also create a managed code callback event that will be used to raise a JavaScript event when the code behind has finished processing. Handling the callback in JavaScript enables us to react to results processed by the Silverlight managed code (which may or may not involve a trip to the server).

#### Example 5. Exposing Silverlight code behind event to JavaScript

```
Page.xaml.cs
using System.Windows.Browser;

namespace SilverlightProject1
{
    [Scriptable]
    public partial class Page : Canvas
    {
        public Page()
        {
            WebApplication.Current.RegisterScriptableObject
("EntryPoint", this);
            MouseLeftButtonDown += Page_MouseLeftButtonDown;
        }

        void Page_MouseLeftButtonDown(object sender, MouseEventArgs e)
        {
            if (CallbackToBrowser != null)
            {
                CallbackToBrowser(this, new EventArgs());
            }
        }

        public void Page_Loaded(object o, EventArgs e)
        {
            // Required to initialize variables
        }
    }
}
```

```

        InitializeComponent();
    }

    [Scriptable]
    public void Test(string Name) {
        //
    }

    [Scriptable]
    public event EventHandler CallbackToBrowser;
}
}

Page.xaml.vb
Namespace SilverlightProject1
    <Scriptable()> _
    Public Partial Class Page
        Inherits Canvas
        Public Sub New()
            WebApplication.Current.RegisterScriptableObject("EntryPoint", Me)
            MouseLeftButtonDown += Page_MouseLeftButtonDown
        End Sub

        Sub Page_MouseLeftButtonDown(ByVal sender As Object, ByVal e
As MouseEventArgs)
            If CallbackToBrowser <> Nothing Then
                CallbackToBrowser(Me, New EventArgs())
            End If
        End Sub

        Public Sub Page_Loaded(ByVal o As Object, ByVal e As EventArgs)
            ' Required to initialize variables
            InitializeComponent()
        End Sub

        <Scriptable()> _
        Public Sub Test(ByVal Name As String)
            '
        End Sub

        <Scriptable()> _
        Public Event CallbackToBrowser As EventHandler

    End Class
End Namespace

TestPage.html.js
<script>

```

```

function CallTest()
{
    var control = document.getElementById("SilverlightControl");
    control.Content.EntryPoint.Test("Valio");
    control.Content.EntryPoint.CallbackToBrowser = onManagedCallback;
}

function onManagedCallback(sender, args)
{
    alert("JS!");
}
</script>

```

### Integration with ASP.NET AJAX Futures

If you are an ASP.NET developer, you should watch the ASP.NET AJAX Futures releases. The Futures releases include early experimental versions of features currently being considered for future versions of ASP.NET and the .NET Framework. The ASP.NET Futures May 2007 release introduced two controls focused entirely on Silverlight: Media and XAML.

The ASP.NET media server control enables you to easily integrate media sources like audio (*.wma* or *.mp3*) and video (*.wmv*) into your web application without requiring any knowledge of XAML or JavaScript. The control is based entirely on JavaScript, so it is currently implemented for the Silverlight 1.0 platform.

The XAML server control enables you to easily integrate XAML and any supporting code (a managed-code assembly, a managed dynamic-language script module, or client JavaScript libraries) into your web application. This control targets the Silverlight 1.1 release with its support for managed code, but it can be used with Silverlight 1.0, too.

The declaration of the XAML control is simple, too. Point the `XamlUrl` property at a XAML file and it will add it to the page, like this:

HTML

```
<asp:xaml runat="server" XamlUrl="simple.xaml"></asp:xaml>
```

More info on the Futures release can be found at <http://ajax.asp.net/futures/>.

### Silverlight 1.1 and WebServices

The new wave of Rich Internet Applications uses different technologies such as AJAX, Flash, and Silverlight to provide their rich client experience. As a side effect, the separation of presentation markup and code from data is stronger than in "classic" web applications. RIAs have clearly defined frontends and back ends (usually web services) that pump data in to the rich client application.

The current alpha release of Silverlight 1.1 has some limitations when it comes to communications with the server, but many of these problems will be addressed in future Silverlight 1.1 builds. For now, Silverlight 1.1 alpha supports web service communication with the web server, though only JSON formatted data is supported. In this section, we'll take a detailed look at how to use web services to connect a Silverlight application to server-side data.

## Using ASMX Web Services

One of the most common ways for ASP.NET developers to get data into a Silverlight application is ASMX Web Services. But because only JSON-formatted result sets from web services are supported in the current Silverlight alpha, you must change the way your ASMX service behaves to use it with Silverlight. To do that, remove the traditional ASMX handler in your application's *web.config* file and add the new `ScriptHandlerFactory` that was introduced by the ASP.NET AJAX extensions, like this:

XML

```
<httpHandlers>
  <remove verb="*" path="*.asmx"/>
  <add verb="*" path="*.asmx" validate="false"
        type="System.Web.Script.Services.ScriptHandlerFactory,
        System.Web.Extensions, Version=2.0.0.0, Culture=neutral,
        PublicKeyToken=31BF3856AD364E35"/>
</httpHandlers>
```

In addition to these *web.config* changes, you must mark the ASMX service class and its public callable methods with the `ScriptServiceAttribute` and the `ScriptMethodAttribute` to use them with Silverlight. We'll examine these attributes in more detail in the POX web service section.

Finally, when your *web.config* has been updated and your service class decorated with the necessary attributes, you must create a proxy to call the web service in Silverlight using the *slwsdl.exe* tool. Visual Studio 2008 simplifies this process by allowing you to add a reference to your web service in a Silverlight project and then auto-generating the proxy. If you choose the manual approach, point the tool at your ASMX file and run it with the `/SilverlightClient` option to generate the proxy file.

## Using POX (Plain Old XML) web service

If you do not want to use an ASMX web service to deliver data to your Silverlight application, you can easily create a POX service using Visual Studio 2008. POX web services are *not* a WSDL breed of web services like the default ASP.NET ASMX services, so they are a simpler approach to service messaging.

The keys to creating a POX service are a couple of new attributes that are available in VS 2008. First, you need to mark your Web Service class with the following attribute:

```
[System.Web.Script.Services.ScriptService]
```

Once that is complete, mark all methods that will be exposed through the service API with this attribute:

```
[ScriptMethod(UseHttpGet=true)]
```

Together, these attributes allow the service methods to be accessed by scripts, and they also notify the web service factory handler that these methods will be available through HTTP GET requests.

Finally, to have a functional service you must enable the GET and POST HTTP commands by manually updating your application's *web.config*. To do that, simply add the following section:

XML

```
<webServices>
  <protocols>
    <add name="HttpGet"/>
    <add name="HttpPost"/>
  </protocols>
</webServices>
```

Now that the POX web service has been created and configured, you need to write code to consume the service in a Silverlight application. Unlike the ASMX service, there are no automatic tools for creating POX service proxies, so the data will have to be manually retrieved, like this:

C#

```
public object GetData()
{
    BrowserHttpRequest request = new BrowserHttpRequest(new Uri(
        "http://localhost/datapump/testService.asmx/GetData?input=last"));

    HttpWebResponse httpResponse = request.GetResponse();
    StreamReader rawReader = new
        StreamReader(httpResponse.GetResponseStream());
    string response = rawReader.ReadToEnd();

    XmlReader reader = XmlReader.Create(new StringReader(response));
    reader.ReadToFollowing("string");
    reader.Read();

    return reader.Value;
}
```

VB

```
Public Function GetData() As Object
    Dim request As New BrowserHttpRequest(New
        Uri("http://localhost/datapump/testService.asmx/GetData?input=last"))

    Dim httpResponse As HttpResponseMessage = request.GetResponse()
    Dim rawReader As New StreamReader(httpResponse.GetResponseStream())
    Dim response As String = rawReader.ReadToEnd()

    Dim reader As XmlReader = XmlReader.Create(New StringReader(response))
    reader.ReadToFollowing("string")
    reader.Read()

    Return reader.Value
End Function
```

And that's all it takes! You can now easily stream data into your Silverlight application from your web server. Simply invoke the service, get the data, and update your application at runtime.

## Solving the cross-domain problem

Silverlight's security model intentionally blocks cross-domain access, so unless the services that you query are on the server that served the page with the Silverlight control, access will not be granted. From a security point of view, this decision makes complete sense, but it also makes development with the Silverlight 1.1 alpha difficult. Nonetheless, the behavior is important because it prevents malicious Silverlight controls from running in the browser. And while we can work around the issue on the server by using a proxy, a more elegant solution is possible with client-side "proxy-like" behavior using JSONP.

JSONP is an unofficial protocol designed to enable cross-domain calls that return JSON from the server. It takes advantage of the browser's security model, which allows references to JavaScript resources at any location by dynamically adding a script element to a page with the `src` property set to a cross-domain URL. If that URL recognizes the JSONP request format, it will return JSON data to a provided callback function name that is executed when the page loads. When the JavaScript callback function executes on page load, the arguments will contain the JSON data from the remote server. It is a bit of a hack, but it is one of the more popular methods for enabling cross-domain data access from the client.

## Using the Astoria project for data access

The goal of Microsoft's project Astoria is to enable backend applications to expose data as services that can be consumed by remote clients, both on local networks and across the Internet. The data services are REST HTTP-based, and URI-like



paths are used to query the data store. You can learn more about project Astoria at <http://astoria.mslivelabs.com/>. The latest May CTP release includes an API and a client library specially designed for Silverlight 1.1. To use these tools with your Silverlight 1.1 projects, extract the *Microsoft.Astoria.SilverlightClient.dll* assembly and add a reference to it from your Silverlight project in Visual Studio.

## LINQ and Silverlight

LINQ, or Language Integrated Query, is a methodology that simplifies and unifies the implementation of data access. Data managed by a program can belong to many different data domains, such as arrays, objects, XML documents, and databases, each with its own specific access model. For example, when you have to query a database, you typically use SQL. You navigate XML data with DOM or XQuery. You use specific APIs to access other data domains. In the end, you have many different programming models to access many different data sources. LINQ tries to solve this issue, leveraging commonalities between the operations in these data models to make data access code consistent regardless of the data's source.

LINQ relies heavily on new language enhancements in C# 3.0 and is available in the Silverlight platform. LINQ will play a significant role in the development of Silverlight applications, as it will be a key technology for querying data returned from the server.

More information on LINQ can be found on MSDN at <http://msdn2.microsoft.com/en-us/netframework/aa904594.aspx>.

## Silverlight 1.1 Downloader object

This is a special-purpose built-in object that provides the ability to download content, such as XAML, JavaScript, or media assets from the web server. It provides functionality for initiating data transfer, monitoring its progress, and retrieving the downloaded content. The object implements behavior that resembles the Ajax paradigm: it can be used for partial updates of the page and transfer of data to and from a web server using HTTP protocol. The Silverlight team engineered the **Downloader** object to follow closely the **XMLHttpRequest** set of APIs (the core communication APIs involved in Ajax). As with other Silverlight security restrictions, the **Downloader** object can only be used to retrieve resources from the domain that originally served the Silverlight page.

With the **Downloader**, you can download single files or whole packages (in other words, zip archives). Using the **Downloader** object in Silverlight is very simple. The following code example shows the necessary code to instantiate a **Downloader** object, begin an asynchronous call, and download an image from the server.

```

C#
private Downloader downloader;
public void InitiateDownload()
{
    downloader.open("GET",
        new Uri("http://www.telerik.com/images/editor/partners/green.png",
UriKind.Absolute));

    downloader.Send();
}

```

```

VB
Private downloader As Downloader
Public Sub InitiateDownload()
    downloader.open("GET", New Uri("http://www.telerik.com/images/
editor/partners/green.png", UriKind.Absolute))

    downloader.Send()
End Sub

```

Because the process is asynchronous, you must register a callback handler to listen to the `Completed` event and provide the required application logic, like this:

```

C#
void downloader_Completed(object sender, EventArgs e)
{
    image.SetSource(downloader, null);
}

```

```

VB
Sub downloader_Completed(ByVal sender As Object, ByVal e As EventArgs)
    image.SetSource(downloader, Nothing)
End Sub

```

Note the second parameter in the `SetSource` method: it is required when the downloaded file is a zip archive, and defines the file inside the archive to be extracted. For example, if the downloader had been used to retrieve a zip archive with images, the *"green.png"* image could be accessed with code like this:

```

C#
image.SetSource(downloader, "green.png");

VB
image.SetSource(downloader, "green.png")

```

You can monitor the download progress by subscribing to the `DownloadProgressChanged` event and checking the `DownloadProgress` property.

# Building Applications with Silverlight 1.1

Now it is time to pull all of the concepts, code samples, and Silverlight 1.1 ideas together to build a sample Silverlight application. In this section, two sample Silverlight controls will be built that can be used in any Silverlight 1.1 application: a custom `button` control and an `upload` control. These controls will illustrate the concepts covered in this Short Cut and provide more insight into the Silverlight application development process.

## Custom Silverlight 1.1 Button Control

Creating a custom Silverlight control involves two basic tasks—defining the visual appearance of the control and providing it with functionality. The UI is primarily defined in XAML markup and the object model is primarily defined in code. Each custom control is described by its API (Application Programming Interface), which exposes public properties, data fields, and events.

The general structure of a custom Silverlight control is a XAML file and a code behind file contained in a Silverlight User Control project.



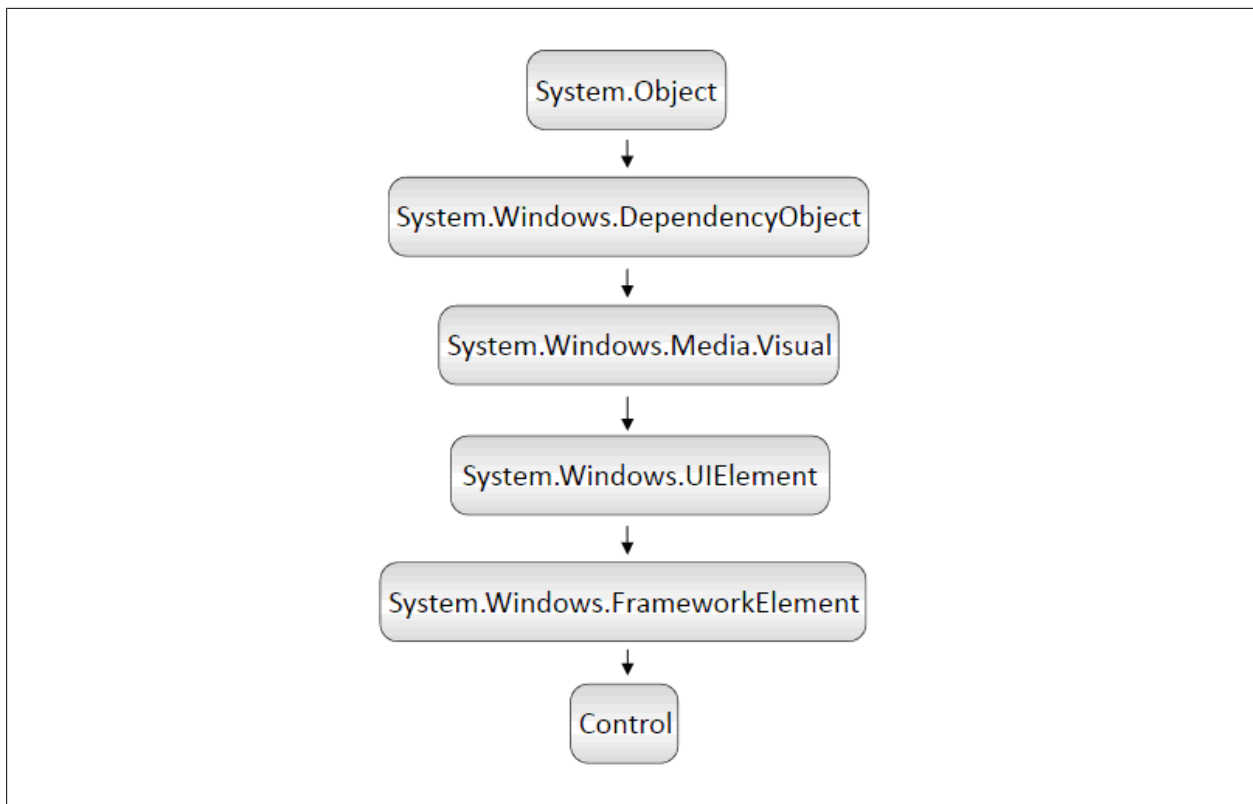
Every custom Silverlight control must inherit from the `.NET Control` class, illustrated in [Figure 25](#).

Our XAML file will contain one `Rectangle` that will determine the boundaries of the control and one `TextBlock` that will serve as a text holder. The XAML markup for the control can be seen in [Example 6](#).

### Example 6. Button control XAML markup

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Width="100"
        Height="50"
        Background="Transparent"
        Cursor="Hand"
        >

    <Rectangle x:Name="buttonBackground"
              Width="100" Height="50"
              Fill="Green" Stroke="Black" StrokeThickness="2"
              RadiusX="10" RadiusY="10"/>
```



**Figure 25. Class hierarchy of .NET Control class**

```

<TextBlock      />
    x:Name="buttonName"
    Canvas.Left="20" Canvas.Top="10"
    FontSize="20" Text="Button" TextWrapping="NoWrap"
  />
</Canvas>

```

From the code above, we should get something similar to the following on our Silverlight page.



With the visual appearance of our button complete, we now need to add its functionality. The basic structure of the code behind of our custom control is like the following:

```

C#
namespace CustomControl_Button
{
    public class MyButton : Control
    {

```

```

        private Canvas root; // root canvas that holds all the xaml elements
    public MyButton()
    {
        //read button's visual appearance from MyButton.xaml
        System.IO.Stream s = this.GetType().Assembly.GetManifestResourceStream
("CustomControl_Button.MyButton.xaml");
        this.root = (Canvas)this.InitializeFromXaml(new System.IO.StreamReader(s)
        .ReadToEnd());
    }
}

```

*VB*

```

Namespace CustomControl_Button
    Public Class MyButton
        Inherits Control
        Private root As Canvas
        ' root canvas that holds all the xaml elements
        Public Sub New()
            'read button's visual appearance from MyButton.xaml
            Dim s As System.IO.Stream =
                Me.[GetType]() .Assembly.GetManifestResourceStream
("CustomControl_Button.MyButton.xaml")
            Me.root = CType(Me.InitializeFromXaml(New
                System.IO.StreamReader(s).ReadToEnd()), Canvas)
        End Sub
    End Class
End Namespace

```

Summarized in few words, our code uses a **Stream** object that reads the entire XAML file and assigns it to a **Canvas** object. In addition to this code, though, we will also need a couple more data fields—a **Rectangle** object and a **TextBlock** object.

*C#*

```

private Rectangle buttonBackground;           // background rectangle
private TextBlock buttonText;                 // button's text
private const int leftIndent = 12;            // the space between the last
                                              // character of the button's text
                                              // and the right border

```

```

public event MouseEventHandler Click;          // mouse click event handler

```

*VB*

```

Private buttonBackground As Rectangle         ' background rectangle
Private buttonText As TextBlock               ' button's text
Private Const leftIndent As Integer = 12      ' the space between the last
                                              ' character of the button's text
                                              ' and the right border

Public Event Click As MouseEventHandler       ' mouse click event handler

```

In the constructor for our `Button` control, we need to attach each of the data fields to a corresponding element in the XAML code. This means that we need to connect `buttonBackground` `Rectangle` with the XAML `Rectangle` named "buttonBackground" and connect the `buttonText` `TextBlock` with the XAML `TextBlock` named "buttonName". This can easily be done by using the `FindName` method. It gets any object in the Silverlight object hierarchy by referencing the object's `x:Name` attribute value.

C#

```
public MyButton()
{
    //read button's visual appearance from MyButton.xaml
    System.IO.Stream s =
        this.GetType().Assembly.GetManifestResourceStream("CustomControl_
        Button.MyButton.xaml");
    this.root = (Canvas)this.InitializeFromXaml(new
        System.IO.StreamReader(s).ReadToEnd());

    //assign the data fields to the elements from MyButton.xaml
    this.buttonBackground = this.root.FindName("buttonBackground") as Rectangle;
    this.buttonText = this.root.FindName("buttonText") as TextBlock;

    //update layout
    this.UpdateLayout();

    //subscribe for MouseLeftButtonUp event
    this.MouseLeftButtonUp += new MouseEventHandler(HandleClick);
}
```

VB

```
Public Sub New()
    'read button's visual appearance from MyButton.xaml
    Dim s As System.IO.Stream =
        Me.[GetType]().Assembly.GetManifestResourceStream("CustomControl_
        Button.MyButton.xaml")
    Me.root = CType(Me.InitializeFromXaml(New
        System.IO.StreamReader(s).ReadToEnd()), Canvas)

    'assign the data fields to the elements from MyButton.xaml
    Me.buttonBackground = CType(Me.root.FindName("buttonBackground"), Rectangle)
    Me.buttonText = CType(Me.root.FindName("buttonText"), TextBlock)

    'update layout
    Me.UpdateLayout()

    'subscribe for MouseLeftButtonUp event
    AddHandler Me.MouseLeftButtonUp, AddressOf HandleClick
End Sub
```

Each custom control needs to expose an API in order to serve any real purpose. For this example, we want to be able to change the text and the color of the button. To do this, we will expose two properties: a `Text` property and a `BackColor` property. The `Text` property will be used to set the text of the button and the `BackColor` property will be used to set the background color of the button. On a general note, it is important to remember when building custom controls or classes that *every* public property, field, and event is considered to be part of the API. For that reason, be careful when choosing what to make public in your controls or classes, as bad APIs can lead to unexpected results.

*C#*

```
//property to set/get button's text
public String Text
{
    get
    {
        return this.buttonText.Text;
    }
    set
    {
        this.buttonText.Text = value;
        this.UpdateLayout();
    }
}
//property to set/get button's background
public SolidColorBrush BackColor
{
    get
    {
        return (SolidColorBrush)this.buttonBackground.Fill;
    }
    set
    {
        this.buttonBackground.Fill = (SolidColorBrush)value;
    }
}
```

*VB*

```
'property to set/get button's text
Public Property Text() As String
    Get
        Return Me.buttonText.Text
    End Get
    Set
        Me.buttonText.Text = value
        Me.UpdateLayout()
    End Set
End Property
```



```

'property to set/get button's background
Public Property BackColor() As SolidColorBrush
    Get
        Return DirectCast(Me.buttonBackground.Fill, SolidColorBrush)
    End Get
    Set
        Me.buttonBackground.Fill = DirectCast(value, SolidColorBrush)
    End Set
End Property

```

Unfortunately, Silverlight cannot detect when significant changes have occurred in the control's state, so we need to provide a method that will update the content after a change has occurred. To handle that, we will create an `UpdateLayout` method that updates our control's appearance.

```

C#
private void UpdateLayout()
{
    //adjust root's and background's width according
    //to the button's text width
    this.root.Width = this.buttonText.ActualWidth + leftIndent;
    this.buttonBackground.Width = this.buttonText.ActualWidth + leftIndent;

    //place the text in the middle of the button
    this.buttonText.SetValue<double>(Canvas.LeftProperty,
(this.root.Width / 2) - (this.buttonText.ActualWidth / 2));
}

VB
Private Sub UpdateLayout()
    'adjust root's and background's width according
    'to the button's text width
    Me.root.Width = Me.buttonText.ActualWidth + leftIndent
    Me.buttonBackground.Width = Me.buttonText.ActualWidth + leftIndent

    'place the text in the middle of the button
    Me.buttonText.SetValue(Of Double)(Canvas.LeftProperty, (Me.root.Width / 2)
- (Me.buttonText.ActualWidth / 2))
End Sub

```

Also, we need to provide an implementation of the `HandleClick` method, like this:

```

C#
private void HandleClick(object sender, MouseEventArgs e)
{
    if (this.Click != null)
    {
        this.Click(this, null);
    }
}

```

```

VB
Private Sub HandleClick(ByVal sender As Object, ByVal e As MouseEventArgs)
    If Me.Click <> Nothing Then
        Me.Click(Me, Nothing)
    End If
End Sub

```

In order to use the button though, some changes need to be made to the *Page.xaml* in our Silverlight application. First, we need to register a namespace that will later be used to access the button in code. Placing the following line right above the Canvas' Width property will handle that:

```

XAML
xmlns:mycontrol="clr-namespace:CustomControl_Button;assembly=ClientBin/
CustomControl_Button.dll"

```

Now the button is ready to be used. To add the button to a Silverlight page, insert the following lines inside the root Canvas of the *Page.xaml* file.

```

XAML
<mycontrol:MyButton x:Name="mybutton" Text="I am a button" BackColor="Gray"
Click="ChangeLayout" />

```

You can then reference the button in the code behind file of *Page.xaml*, like this:

```

C# (Page.xaml.cs)
private void ChangeLayout(object sender, MouseEventArgs e)
{
    this.mybutton.Text = "I was clicked";
    this.mybutton.BackColor = new SolidColorBrush(Colors.Red);
}

```

```

VB (Page.xaml.vb)
Private Sub ChangeLayout(ByVal sender As Object, ByVal e As MouseEventArgs)
    Me.mybutton.Text = "I was clicked"
    Me.mybutton.BackColor = New SolidColorBrush(Colors.Red)
End Sub

```

Instead of adding our custom button to the page in our XAML code, it is also possible to add it to the page programmatically. Using the techniques examined earlier in this Short Cut for standard Silverlight objects, we can add our custom button to the page using code in [Example 7](#).

### Example 7. Adding custom button to XAML Canvas programmatically

```

C# (Page.xaml.cs)
public void Page_Loaded(object o, EventArgs e)
{
    // Required to initialize variables
    InitializeComponent();
    MyButton button = new MyButton();
}

```

```

        button.Text = "Created from Code";
        button.BackColor = new SolidColorBrush(Colors.Blue);
        button.Click += new MouseEventHandler(button_Click);

        this.Children.Add(button);
    }

    void button_Click(object sender, MouseEventArgs e)
    {
        ((MyButton)sender).Text = "I was clicked";
        ((MyButton)sender).BackColor = new SolidColorBrush(Colors.Red);
    }

```

*VB (Page.xaml.vb)*

```

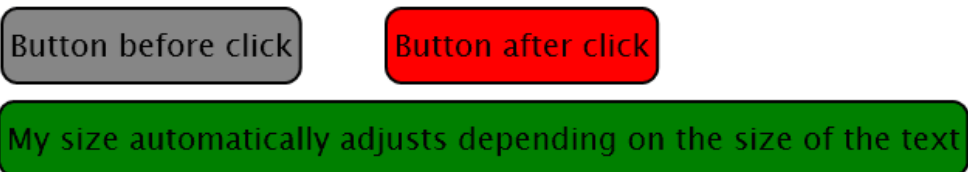
Public Sub Page_Loaded(ByVal o As Object, ByVal e As EventArgs)
    ' Required to initialize variables
    InitializeComponent()
    Dim button As New MyButton()
    button.Text = "Created from Code"
    button.BackColor = New SolidColorBrush(Colors.Blue)
    AddHandler button.Click, AddressOf button_Click

    Me.Children.Add(button)
End Sub

Sub button_Click(ByVal sender As Object, ByVal e As MouseEventArgs)
    (DirectCast(sender, MyButton)).Text = "I was clicked"
    (DirectCast(sender, MyButton)).BackColor = New
        SolidColorBrush(Colors.Red)
End Sub

```

If everything worked correctly, your Silverlight application will render a single custom button to the page in the browser. When you click on the button, the background color and text will change.



## Building a Silverlight Upload Control

Now that we are comfortable building custom controls in Silverlight 1.1 alpha, we will dig deeper and create a more sophisticated control—an upload control. The standard file control available in the HTML object model (`<input type="file" ...>`) lacks some very handy features such as:

- Multiple file selection
- Asynchronous upload
- File size validation on the client

In Silverlight, there is `OpenFileDialog` class that provides you with more advanced functionality and can help deliver some of the above features.

Before we begin building our control, let's take a broad look at the steps that need to be completed. First, there should be a module that allows a user to select files from the filesystem. Then the selected files need to be read, validated, and posted to the server. On the server there needs to be an HTTP Handler that saves the posted files to a specific location (it could be a filesystem or database).

Since Silverlight lives on the client, you cannot implement the server-side handler in Silverlight code. The other two modules that take care of selecting and uploading the files can be easily implemented, though, so that's what our control will focus on completing. For this example, these two modules will be called `FileSelector` and `FileUploader`.

Implementing the `FileSelector` module is very easy with Silverlight 1.1. The only thing we need to do is use the `System.Windows.Controls.OpenFileDialog` control and configure it to our needs. Some of the major properties this class exposes are:

- **Filter:** Specifies the extensions filter which should be used in the file dialog.
- **EnableMultipleSelection:** Indicates whether multiple file selection is allowed.
- **Title:** Title of the file dialog window.

This object should be very familiar to Windows Forms developers. ASP.NET developers may not be used to accessing files this way, but the process is standard in WinForms. To implement the `OpenFileDialog` in our Silverlight upload control, we'll use code like this:

```
C#
OpenFileDialog fileDialog = new OpenFileDialog();
fileDialog.EnableMultipleSelection = true;
fileDialog.Filter = "Picture Files (*.jpg; *.png)|*.jpg; *.png";
fileDialog.Title = "Please choose files to upload!";

if (fileDialog.ShowDialog() == DialogResult.OK)
{
    foreach (FileDialogFileInfo file in fileDialog.SelectedFiles)
    {
        // process the selected files
        Stream stream = file.OpenRead();
    }
}
```

```

    }
}

VB
Dim fileDialog As New OpenFileDialog()
fileDialog.EnableMultipleSelection = True
fileDialog.Filter = "Picture Files (*.jpg; *.png)|*.jpg; *.png"
fileDialog.Title = "Please choose files to upload!"

If fileDialog.ShowDialog() = DialogResult.OK Then
    For Each file As OpenFileDialogFileInfo In fileDialog.SelectedFiles
        ' process the selected files
        Dim stream As Stream = file.OpenRead()
    Next
End If

```

Implementing the `FileUploader` module requires a little more work than the `FileSelector` module, but the process remains relatively simple. To get the uploaded file data to the server, we will make use of the `System.Windows.Browser.Net.BrowserHttpRequest`. This is the class that is used to create asynchronous requests to a remote server in Silverlight 1.1. As we've mentioned previously, cross-server requests are not possible in the current alpha release, but there are plans for enabling such requests in the near future.

Configuration and usage of the `BrowserHttpRequest` class is quite simple. To instantiate the class and upload our file data, we'll use the code outlined in [Example 8](#).

### Example 8. Uploading data from Silverlight to the server using `BrowserHttpRequest`

```

C#
// configure the request (serviceUrl is the url of the http handler.
// See below for explanation)
HttpRequest request = new BrowserHttpRequest(serviceUrl);
request.Method = "POST";

// the logic for chunk file upload
byte[] buffer = new byte[bufferSize];
int bytesRead = stream.Read(buffer, 0, bufferSize);
string encodedData = Convert.ToBase64String(buffer, 0, bytesRead);

// configure the stream
StreamWriter writer = new StreamWriter(request.GetRequestStream());
writer.Write(String.Format("f={0}\np={1}\nd={2}", fileName, position,
    encodedData));
writer.Flush();

// post to the server

```

```

request.BeginGetResponse(new AsyncCallback(OnResponse), null);

private void OnResponse(IAsyncResult asyncResult)
{
    // on response - check the status code
    HttpResponseMessage response = request.EndGetResponse(asyncResult);

    if (response.StatusCode != HttpStatusCode.OK)
    {
        throw new WebException();
    }

    response.Close();
    request = null;
}

VB
' configure the request (serviceUrl is the url of the http handler.
' See below for explanation)
Dim request As HttpRequest = New BrowserHttpRequest(serviceUrl)
request.Method = "POST"

' the logic for chunk file upload
Dim buffer As Byte() = New Byte(bufferSize) {}
Dim bytesRead As Integer = stream.Read(buffer, 0, bufferSize)
Dim encodedData As String = Convert.ToBase64String(buffer, 0, bytesRead)

' configure the stream
Dim writer As New StreamWriter(request.GetRequestStream())
writer.Write([String].Format("f={0}" & Chr(10) & "p={1}" & Chr(10) & "d={2}",
    fileName, position, encodedData))
writer.Flush()

' post to the server
request.BeginGetResponse(New AsyncCallback(OnResponse), Nothing)

Private Sub OnResponse(ByVal asyncResult As IAsyncResult)
    ' on response - check the status code
    Dim response As HttpResponseMessage = request.EndGetResponse(asyncResult)

    If response.StatusCode <> HttpStatusCode.OK Then
        Throw New WebException()
    End If

    response.Close()
    request = Nothing
End Sub

```

The nice thing about using StreamWriter is that you can actually upload the file in chunks, which is perfect for optimizing memory usage on the server (especially when big files are uploaded).

We mentioned previously that our Silverlight upload control could only handle the selection of files and uploading of data to the server. We will need one last module, implemented as an HTTP Handler, running on the server to receive the uploaded data and save it on the server. While this is not part of the Silverlight control, we will provide some basic steps for creating the Handler and using it with the Silverlight uploader.

Our Handler needs to accept the incoming data stream from the Silverlight application and save it to a file on server. **Example 9** shows code for a very simple implementation of the HTTP Handler that takes the posted stream and writes it into a file.

### **Example 9. ASP.NET HTTP Handler to receive upload data from Silverlight**

```
C#
using System;
using System.IO;
using System.Web;
using System.Web.UI;
using System.Web.SessionState;
using System.Diagnostics;

namespace Telerik.Silverlight.UI
{
    public class RadUploadHandler : IHttpHandler
    {
        public bool IsReusable
        {
            get
            {
                return false;
            }
        }

        public void ProcessRequest(HttpContext context)
        {
            string fileName = null;
            long position = 0;
            byte[] buffer = null;

            using (StreamReader reader = new
                StreamReader(context.Request.InputStream))
            {
```



```

        string body = reader.ReadToEnd();
        string[] parts = body.Split('\n');

        for (int i = 0; i < parts.Length; i++)
        {
            char key = parts[i][0];
            string value = parts[i].Substring(2);

            switch (key)
            {
                case 'f':
                    fileName = value;
                    break;
                case 'p':
                    position = Convert.ToInt64(value);
                    break;
                case 'd':
                    buffer = Convert.FromBase64String(value);
                    break;
                default:
                    throw new InvalidDataException();
            }
        }
    }

    using (FileStream stream = File.OpenWrite(GetPath(context,
        fileName)))
    {
        stream.Position = position;
        stream.Write(buffer, 0, buffer.Length);
    }
}

private string GetPath(HttpContext context, string fileName)
{
    return context.Server.MapPath("~/Uploads/" +
        Path.GetFileName(fileName));
}
}

```

*VB*

```

Imports System
Imports System.IO
Imports System.Web
Imports System.Web.UI
Imports System.Web.SessionState
Imports System.Diagnostics

```

Namespace Telerik.Silverlight.UI

```

Public Class RadUploadHandler
    Implements IHttpHandler

    Public ReadOnly Property IsReusable() As Boolean
        Get
            Return False
        End Get
    End Property

    Public Sub ProcessRequest(ByVal context As HttpContext)
        Dim fileName As String = Nothing
        Dim position As Long = 0
        Dim buffer As Byte() = Nothing

        Using reader As New
            StreamReader(context.Request.InputStream)

            Dim body As String = reader.ReadToEnd()
            Dim parts As String() = body.Split(Chr(10))

            Dim i As Integer = 0
            While i < parts.Length
                Dim key As Char = parts(i)(0)
                Dim value As String = parts(i).Substring(2)

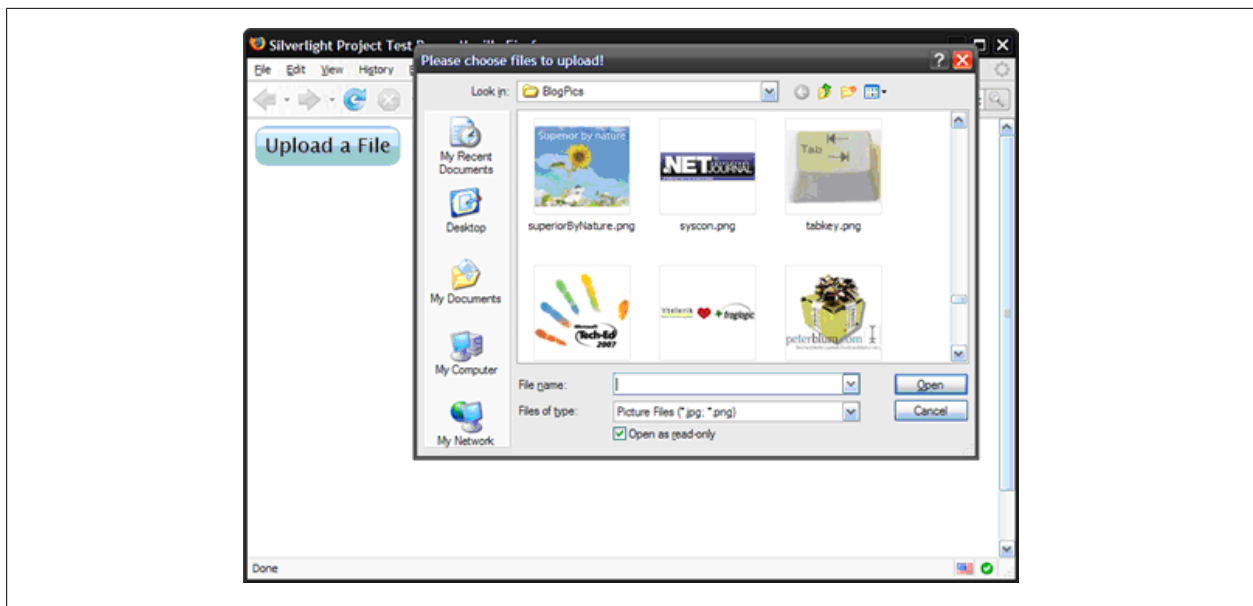
                Select Case key
                    Case "f"C
                        fileName = value
                    Case "p"C
                        position = Convert.ToInt64(value)
                    Case "d"C
                        buffer =
                            Convert.FromBase64String(value)
                    Case Else
                        Throw New InvalidDataException()
                End Select

                i += 1
            End While
        End Using

        Using stream As FileStream = File.OpenWrite(
            GetPath(context, fileName))
            stream.Position = position
            stream.Write(buffer, 0, buffer.Length)
        End Using
    End Sub

    Private Function GetPath(ByVal context As HttpContext,
        ByVal fileName As String) As String

```



**Figure 26. Custom Silverlight 1.1 Upload Control**

```

        Return context.Server.MapPath("~/Uploads/" +
            Path.GetFileName(fileName))
    End Function
End Class
End Namespace

```

After we compile our HTTP Handler into a separate assembly (*Telerik.Silverlight.UI.dll*, in this case), the only remaining task is to register the Handler in the *web.config* file, like this:

In `<configuration>`, `<system.webserver>`, `<handlers>` section of the web config add the following lines:

```

XML
<system.webServer>
  <handlers>
    <add name="RadUploadHandler" verb="POST" path="TestUpload.ashx"
      type="Telerik.Silverlight.UI.UploadHandler,
Telerik.Silverlight.UI"/>
  </handlers>
</system.webServer>

```

To finish this demo, we simply need to add a UI to our Silverlight upload control that can be used to open the file selection dialog. We will do that by rolling the functionality of our upload control into the UI of our custom Button to produce a custom upload control that can be quickly added to any Silverlight XAML markup. The final control logic and markup will produce results like **Figure 26**.

Clearly, building custom controls for Silverlight 1.1 is easy with the built-in .NET support. Classes like *BrowserHttpRequest* simplify tasks that require sending

data to the web server, and Visual Studio 2008 provides all of the tools and IntelliSense required to make programming in the new framework easy.

## Conclusions

Silverlight 1.1 is going to open up a whole new world for .NET developers. Falling somewhere between traditional Windows programming and web programming, Silverlight will attract developers from both disciplines and create a convergence of ideas that has never been experienced in the .NET community.

In its current alpha state, Silverlight 1.1 certainly has a number of issues to address. From cross domain resource access to a more robust control model, there is no shortage of major issues that make it difficult to do full scale development with Silverlight 1.1 at the time of this writing. On top of that, Silverlight 1.1 is not feature-complete. It is easy to forget that this is an *alpha* build with all of the focus on beta products like Silverlight 1.0 and Visual Studio 2008, but make no mistake—this is an alpha and we can expect many changes before it's released.

One thing is certain: when Silverlight 1.1 is released, it will create new opportunities for .NET developers to use their skills to create Rich Internet Applications with unprecedented efficiency. And now you are prepared. This Short Cut has navigated you through the alpha territory, showing you the ins and outs of Silverlight 1.1 with plenty of examples along the way. Whether you adopt Silverlight 1.1 when it arrives or not, you now understand it.

## Resources

This Short Cut has taken you through many aspects of the current Silverlight 1.1 alpha, but in no way does it cover the breadth of the available knowledge on the topic. Excellent Internet resources exist for developers looking to learn more about Silverlight, and among our favorite resources are the following sites and blogs:

### Links

- <http://silverlight.net>
- <http://www.microsoft.com/silverlight/>
- <http://www.telerik.com/silverlight>

### Blogs

- <http://blogs.msdn.com/mharsh/default.aspx>
- <http://weblogs.asp.net/scottgu/>
- <http://silverlight.net/blogs/JesseLiberty>

- <http://adamkinney.com>
- <http://blogs.msdn.com/tims/>
- <http://geekswithblogs.net/WynApseTechnicalMusings/Default.aspx>
- <http://telerikwatch.com>

## Special Thanks

Special thanks to Valentin Stoychev, Dimitar Kapitanov, Kiril Stanoev, Irina Palichorova, Nikolay Nedev, and Hristo Kosev from the Telerik Silverlight Team for their major contributions to this book. Additional thanks to Microsoft evangelists Adam Kinney and Jesse Liberty for reviewing this Short Cut and providing their valuable feedback.