

# Silverlight 1.1 in Action

Chad A. Campbell



Unedited Draft



 MANNING

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=369>



**MEAP Edition**  
**Manning Early Access Program**

Copyright 2007 Manning Publications

For more information on this and other Manning titles go to  
[www.manning.com](http://www.manning.com)

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=369>

# *Table of Contents*

Chapter 1: Introducing Silverlight

Chapter 2: Silverlight: Harmony with the Web

Chapter 3: Managing the Basics

Chapter 4: Handling User Interaction

Chapter 5: Working with Data

Chapter 6: Networking and Communication

Chapter 7: Managing Multimedia

Chapter 8: Getting a Grip on Graphics

Chapter 9: Let's Get Animated

Chapter 10: Skinning

Chapter 11: Content Delivery and Storage

Chapter 12: Share the Light: Distribution and Deployment

# *1 Introducing Silverlight*

You've purchased this book wanting to learn about Microsoft's Silverlight 1.1 platform, so what is it? Silverlight is a client-side runtime that enables you to design, develop, and deliver rich, interactive experiences over the internet. Truly interactive experiences have traditionally been confined to desktop applications. However, unlike the 1.0 version, Silverlight 1.1 provides the power of the .NET framework to help you reach beyond the desktop to deliver these experiences across a variety of browsers, platforms, and devices<sup>1</sup>.

This chapter will lay the groundwork for the rest of this book. In order to prepare for our journey together, we will first discuss what Silverlight offers within the rich interactive application space. Then we will examine the designer/developer collaboration advancements allotted by Silverlight. These collaboration advancements are enabled by an XML-based language called XAML, which we will also cover. In order to begin, let's raise the sails on our catamaran and begin on this fast-paced journey.

## *1.1 Silverlight Up Your Life*

The winds of excitement are whipping around the World Wide Web as individuals increasingly rely on the internet for their software needs. When the internet began, it was primarily used for sharing static content. However, as web adoption exploded, users gradually desired more pizzazz through their web browsers. They wanted to perform their banking and shopping through the web. They wanted to use their computer to share pictures, songs, and videos to strengthen existing friendships and build new ones. Users wanted all of this and more in a rich and responsive manner.

The technologies created for delivering static content quickly became constrictive when it came time to deliver these more dynamic experiences. The web development community attempted to meet these challenges by cobbling together a multitude of technologies that tended more towards wizardry than recommend development practices. JavaScript emerged as a fundamental contrivance for these magicians to perform their tricks. And skilled web developers navigated the HTML document object model (DOM) much like a safari leader wielding a small dagger within a treacherous jungle.

The jungle of technologies thickened as a variety of browser-based plug-ins arose to address the needs of the user interface. However, often times these solutions were, and are, incomplete. They often overlook the importance of a true development experience. But Silverlight doesn't. Silverlight delivers the three tools that enable us

---

<sup>1</sup> Silverlight content has been shown to run on Windows Mobile 6 devices. However, this book will focus on developing Silverlight content within a browser because the timeframe for Silverlight on the Windows Mobile platform has not been announced.

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=369>

to slash through this overgrown jungle. These tools are represented as the three P's of Silverlight: Productivity, Performance, and Portability.

### ***1.1.1 Productivity***

Imagine how painful grocery shopping would be if only one store sold bread, and another sold only milk, and yet another sold only meat. We haven't even discussed cereal, toilet paper, and other day-to-day items. But, as you can imagine, grocery shopping would be a frequent source of frustration in your life. Strangely, web development often times imitates this analogy.

Silverlight addresses the inherent difficulties of traditional web-based software by unifying 2D graphics, animation, input, media, and text into a single application programming interface (API). This API gives us the flexibility to create networked applications that seamlessly integrate with data and services. Integration with data and services assists us in overcoming the disconnected nature of the internet. This in turn allows us to deliver a more responsive experience in a more productive manner.

A robust, powerful development experience is critical for creating valuable solutions. It allows us to create products more quickly by allowing us to focus on solutions instead of problems. It enables a greater degree of flexibility, which breeds innovation. It allows us to take advantage of previous development endeavors by minimizing integration efforts. It encourages robustness through more efficient testing mechanism. Silverlight enables all of this, and more, because when it comes to web-based software, it has a unified development story that is second-to-none.

This development story is incredibly powerful. Often times though, powerful tools are large and unwieldy; they tend towards oafish, clumsy behavior. These adjectives generally depict a decrepit, world-renowned giant in fairytales. But Silverlight isn't any of these things. In fact, it is an incredibly nimble and performant platform that allows us to create highly responsive and performant solutions.

### ***1.1.2 Performance***

Getting stuck in the quicksand of long response times can be a frustrating experience. Long response times within an application can cause us to squirm and fidget in attempt to remove the pain-inducing wait times. These attempts ultimately cause us to sink further into despair and frustration. Whether we are developing an exciting entertainment experience, or a line-of-business application that will be used for hours on end, performance *is* important.

As we will see shortly, performance flows through Silverlight's veins like a Triple Crown winning thoroughbred. Silverlight outpaces similar technologies so we can light up the jungle with highly responsive user experiences that are both calming and engaging. These calming experiences can help us rise up out of the quicksand pits associated with web-based applications and run towards a responsive web.

In order to separate the ponies from the horses, Alexey Gavrilov put together a series of animations that compares various web-focused frameworks. Each animation involves animating 16 balls bouncing around a box and calculating the frames-per-

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=369>

second (fps). As the following illustration shows, Silverlight clearly outperforms the rest.

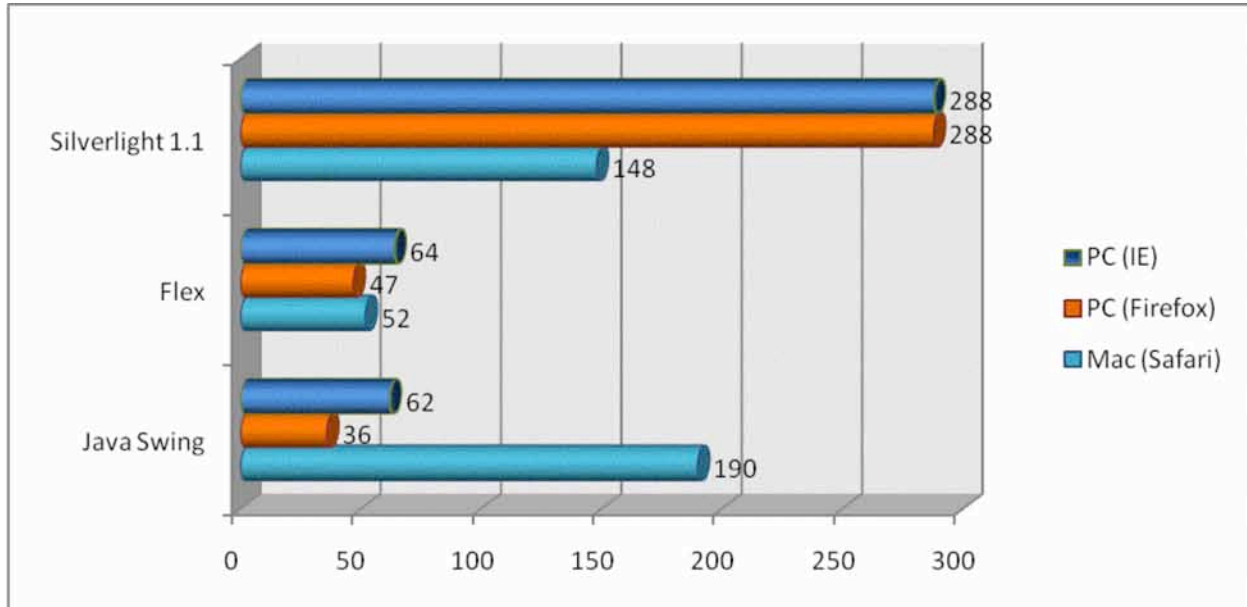


figure 1.1 Calculated frames per second, per technology, per platform, and per browser<sup>2</sup>.

As figure 1.1 illustrates, Silverlight can perform magnitudes more efficiently than similar technologies. However, these performance advantages are *not* limited to the Microsoft Windows platform. Instead, Silverlight goes beyond expectations, and the Microsoft platform, to give us freedom through portability.

### 1.1.3 Portability

Silverlight is a cross-browser, cross-platform user experience juggernaut that was designed with portability in mind. Before the official name of Silverlight was announced, it was publicly known as “Windows Presentation Foundation Everywhere” or “WPF/E for short. This is because Silverlight uses a subset of the Windows Presentation Foundation (WPF) specification that can be delivered beyond the Windows platform.

Those interested in Silverlight have often asked me, what does “everywhere” really mean? Well, natively, Silverlight content has the capacity to run on approximately 96% of the computers on the internet. This is based on the fact that Silverlight has been built and tested to run on specific operating systems and browsers.

Silverlight, like most software, must run on an operating system. Silverlight natively supports the Microsoft Windows *and* Apple Macintosh platforms. In addition, Microsoft is in partnership with Novell to provide a Silverlight implementation for Linux through the “Moonlight” project. The following chart shows the list of operating systems, and their market share, that Silverlight supports (Net Applications, 2007).

<sup>2</sup> The Internet Explorer and Firefox statistics were compiled using a freshly formatted Dell Inspiron 6000 with an Intel Pentium M 2.00 GHz processor, 1GB of RAM, and Windows XP (SP2) installed. The Safari statistics were compiled using an Apple MacPro 2.66 GHz DuoCore processor with 2GB of RAM. These tests, along with source code, are available at <http://www.bubblemark.com>.

Please post comments or corrections to the Author online forum at <http://www.manning-sandbox.com/forum.jspa?forumID=369>

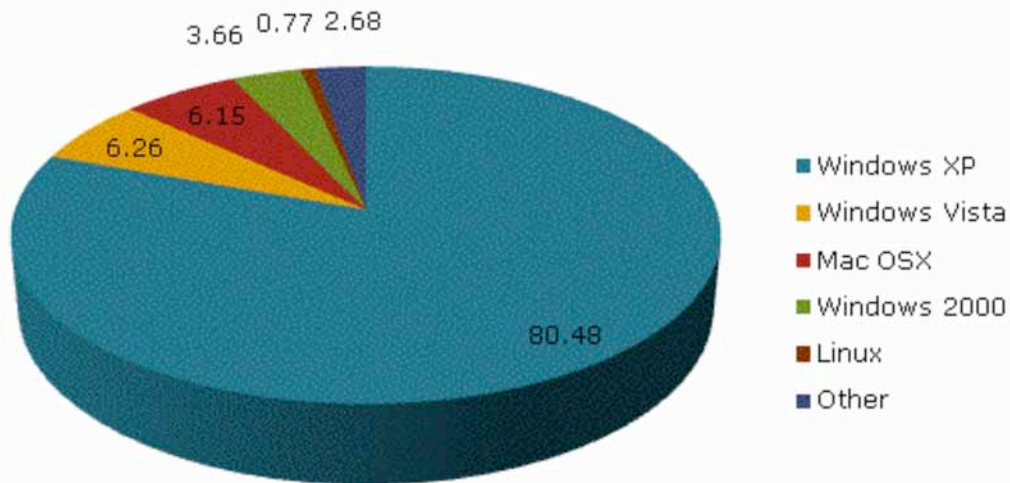


figure 1.2 Silverlight supports the platforms listed. The “other” portion represents the platforms that Silverlight does not support.

The specific operating systems listed in figure 1.2 account for approximately 97% of the operating systems used on the internet. The remaining 3% represents the operating systems that are not supported by Silverlight.

Because Silverlight is a browser plug-in, it must run within the security sandbox of an internet browser. Silverlight supports the most popular web browsers by addressing 96% of the browsers used in the market. In addition to Internet Explorer, Silverlight also supports the Firefox, Opera, and Safari web browsers. In addition, the Moonlight project extends support to the Konqueror browser on Linux. The following chart shows the market share of these browsers

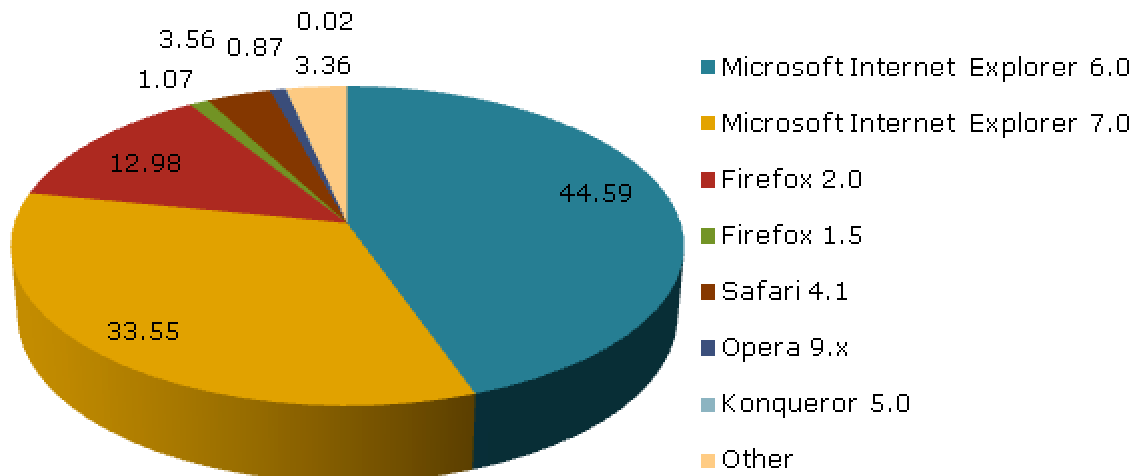


figure 1.3 Silverlight content can run within the browsers and versions listed within the chart. The “other” portion represents the browsers that are not supported by Silverlight or Moonlight.

Please post comments or corrections to the Author online forum at <http://www.manning-sandbox.com/forum.jspa?forumID=369>

The browsers explicitly listed in figure 1.3 are supported by either the Silverlight or Moonlight runtimes. As this chart, and the one in figure 1.2 illustrates, Silverlight can reach approximately 96% of the computers on the internet.

At this point, it's clear to see that Silverlight is a wildfire burning up the jungle of technologies associated with the traditional web. The performance and productivity characteristics make sure Silverlight a wise choice for developing rich interactive applications. The cross-browser, cross-platform reach of Silverlight make it the responsible choice. There is one more productive advantage that is SOOOO big; it requires its own section: The designer/developer collaboration.

## 1.2 Why Can't We Be Friends?

Traditionally, the tools and technologies involved in web-based software development have separate designers and developers like the Berlin Wall separated East and West Germany. Designers would utilize powerful illustrative toolsets to create vivid, intricate user interface designs in the form of a mockup. The mockup would then be tossed over the great wall and developers would dissect it and extract the necessary images in an effort to re-create the visual depictions.

Despite the most committed efforts, these designs would occasionally stretch beyond the capabilities of traditional web standards. This would then force a compromise between design and development that would often result in selecting the lowest common denominator. However, if we review the tiers associated with a webpage, as shown in the following illustration, we begin to see that the apartheid between designers and developers doesn't need to exist.

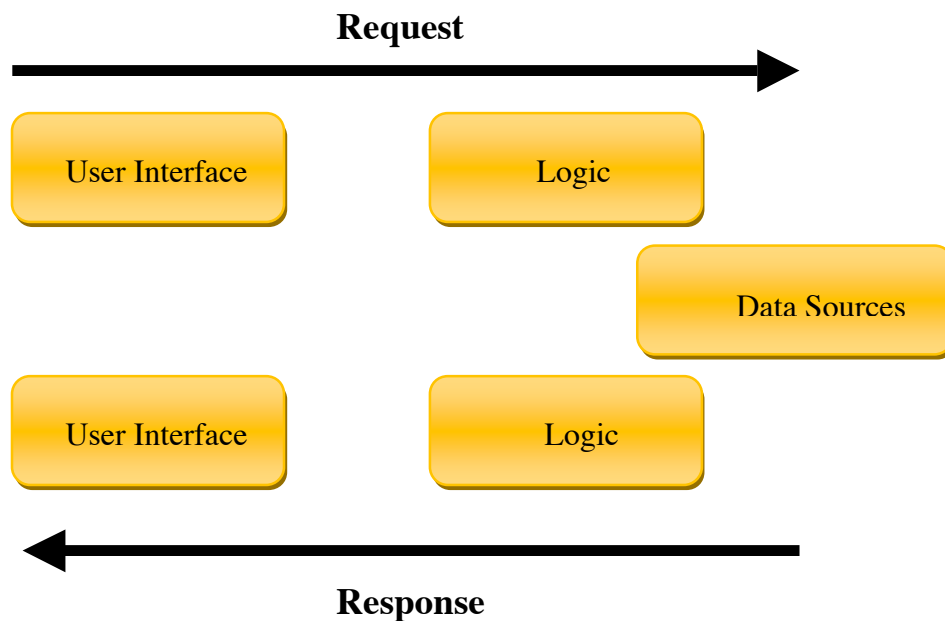


figure 1.4 The tiers of a webpage.

Please post comments or corrections to the Author online forum at <http://www.manning-sandbox.com/forum.jspa?forumID=369>

The illustration in figure 1.4 clearly shows a separation between the code (represented by the “Logic” and “Data Sources” blocks) and the design (represented by the “User Interface” blocks). If we recall the role definitions mentioned in the preface, we will remember that designers are generally very skilled at creating intricate user interfaces. We will also recall that developers generally possess strong coding and architecting skills. As figure 1.4 illustrates, there is a clear separation between the tasks associated with these roles. In fact, if we revisit the previous illustration and add the designer and developer roles to the illustration, we will get something that looks like the following:

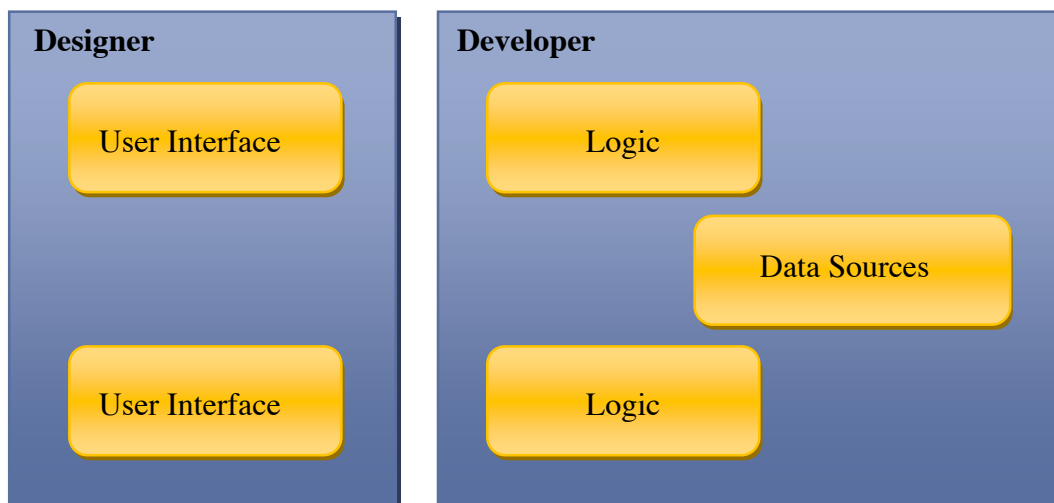


figure 1.5 Designer / Developer work distribution.  
Figure

As figure 1.5 clearly shows, there is an undeniable separation between the designer and developer roles. This separation seemingly mimics the segregation we are attempting to avoid. So how do we bridge these two worlds? This is the role of an XML-based language called XAML.

The details of XAML are covered in section 1.3. For now though, it is important to understand that XAML acts as a bridge between our design and our code that makes crossing this divide child's play. This enables a refreshing collaborative experience between designers and developers that allows us to create valuable software solutions in shorter amounts of time. By keeping our design separate from our code, we can take advantage of tools targeted for specific tasks, while at the same time working in a manner that we are used to.

Most of us are probably accustomed to writing software without a designer. XAML gives us the flexibility to work with a designer as much, or as little, as necessary. Regardless of your development approach, it is still important that we build a firm understanding of the “User Interface” bubbles in figure 1.5. By fostering this understanding, we can ensure that we can proficiently code to the user interface or go as far as creating the entire user interface ourselves. With that said, figure 1.5 may be more appropriately depicted as the following:

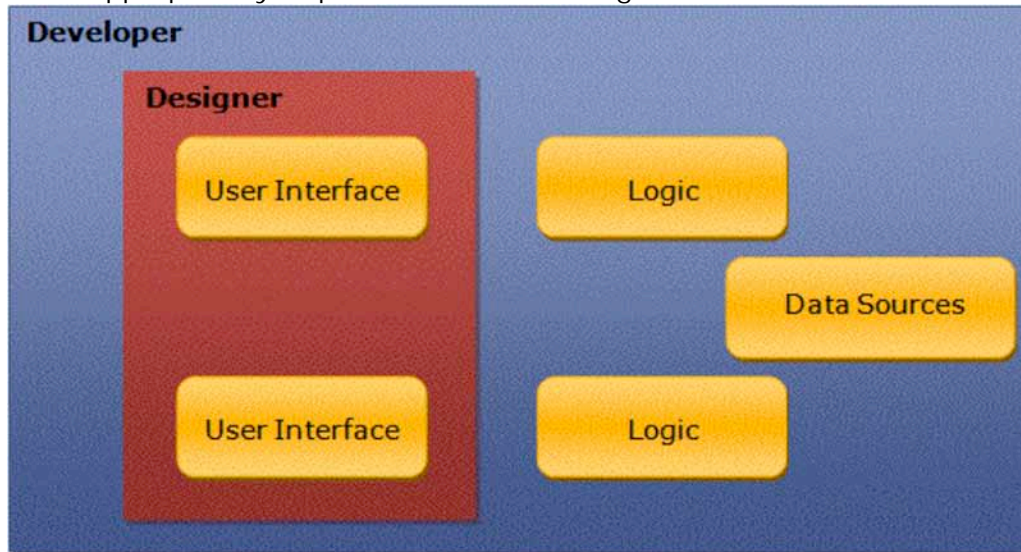


figure 1.6 – Another Designer/Developer work distribution viewpoint

Figure 1.6 clearly illustrates the breadth and depth of knowledge required by a Silverlight developer. This visualization makes it easier to connect with the developer and designer experiences available within Silverlight.

### ***1.2.1 – The Developer Experience***

So, what makes the development experience in Silverlight so special? One word: .NET (or is it two words “dot NET”).

Silverlight 1.1 contains a full-featured version of the .NET CLR along with a trimmed-down version of the framework class library that enables us to easily create rich and connected applications across platforms. This version of the framework is specific to Silverlight 1.1; however it still includes all of the powerful presentation, data, networking, and other programming features we expect as developers. It simply omits features such as com-interop that are irrelevant in web application development.

As an additional bonus, this powerful version of the framework is backed by some of the industry's leading tools, including Visual Studio. Visual Studio is probably the most widely-recognized tool when it comes to .NET development. With an established collection of world-class tools, we can rely on the fact that our time will be spent on solutions instead of annoyances.

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=369>

Typically, integration between tools can be an annoying problem that can make collaboration frustratingly difficult. However, Visual Studio allows a Silverlight template that seamlessly integrates the developer and designer worlds, thus promoting collaboration. This collaboration enables us to easily deliver a highly stylized Silverlight application, thanks in part to the terrific designer experience available within Microsoft Expression Blend.

### ***1.2.2 – The Designer Experience***

Microsoft Expression Blend, or simply Blend, is a professional design tool that empowers designers to create rich, vivid user experiences. Though Blend allows you to create rich Windows-based applications, it also enables you to create rich, connected web-based applications with Silverlight. Essentially, Blend can be used to simplify the process of incorporating blurs, gradients, and other complex design elements, such as animations, within your Silverlight applications.

The significance of Blend in relation to Silverlight lies in two riveting details. First, as intricate design elements are constructed within Blend, the XAML for these elements is automatically created for you. Secondly, the Silverlight project structure schema within Blend is consistent with the project schema in Visual Studio. This allows for frictionless collaboration between designers and developers, or enables you to use both tools simultaneously on your own computer.

As depicted, this valuable tool is worth familiarizing ourselves with; in addition, Microsoft has provided a free trial version of Blend through their website, so there really is no reason not to use it with this book. It should be mentioned that Blend is not required for Silverlight development; however we will be using it through some of the optional “Blending It Together” walkthroughs in this book. On the other hand, unlike Blend, XAML *is* virtually required for Silverlight development.

## ***1.3 XAML***

The eXtensible Application Markup Language (XAML) is a declarative language that enables us to initialize objects in XML format. This format enables us to easily visualize a hierarchy of elements while at the same time separating content from code. This is possible because each XAML element maps to a .NET type. While at the same time, each attribute within an element corresponds to a property within a .NET type. This is probably best illustrated as the following:

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=369>



figure 1.6 Any object that we create in XAML can also be created within code.

The illustration in figure 1.6 shows three code equivalents to a segment of XAML. You will notice that the `TextBlock` element in the XAML code corresponds to an initialization statement within the code segments. This occurs because each time an element is created in XAML; the corresponding .NET type's default constructor is called behind the scenes. Also occurring behind the scenes, is the executable code that is defined within the code-behind files.

### 1.3.1 Code-Behind

Much like ASP.NET, XAML pages support the concept of code-behind pages. Code-behind pages enable us to separate code from design, by placing the UI related code in the XAML and the executable code within a linked source file. This relationship between XAML and source code is best exemplified by the following illustration.

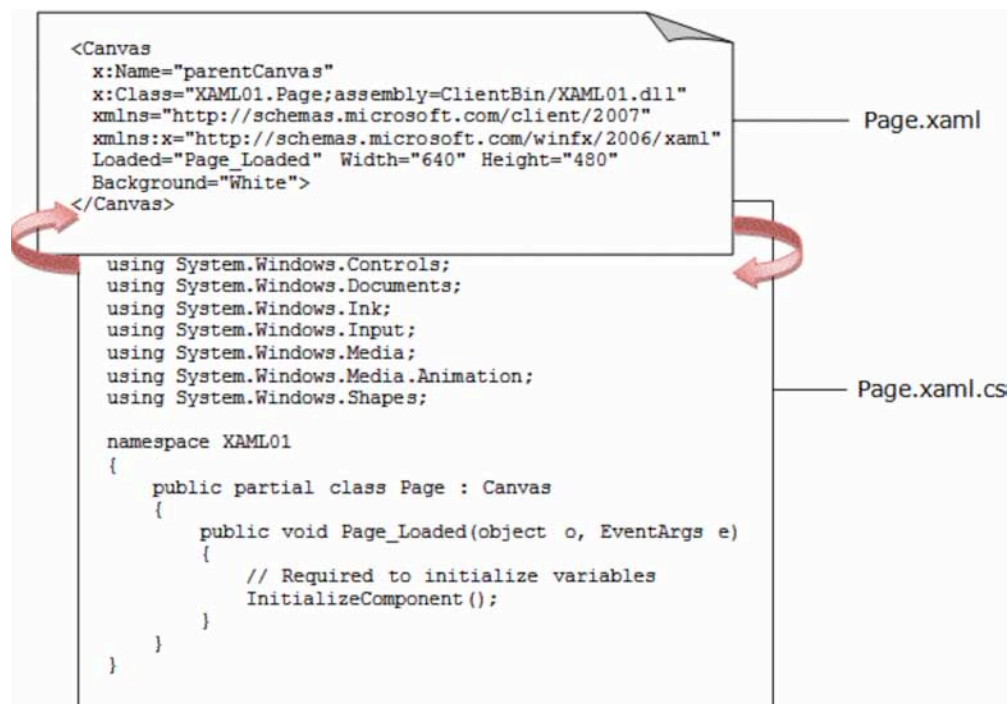


figure 1.7 The relationship between a XAML file and a Code-Behind file within a C# project.

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=369>

As figure 1.7 illustrates, the XAML code is stored within a .xaml file while the code-behind class definition is stored within a .xaml.cs (the extension will vary based upon what language you are using) file. In all honesty though, the class definition is stored within two files, a file with the .xaml.cs suffix and a file with the .xaml.g.cs suffix which is automatically created by Visual Studio. While the definition for the `InitializeComponent` method from figure 1.7 is stored within the .xaml.g.cs file, all of our coding efforts should take place within the .xaml.cs file and for this reason, we will focus on that file.

The XAML file references the code-behind file through the `x:Class` attribute. This class definition is compiled and stored within an assembly that gets placed within a directory relative to the application called "ClientBin". You will notice from figure 1.7 that the path to the assembly file is also provided within the `x:Class` attribute in addition to the class name.

The class definition is primarily used to handle the events triggered through the user interface defined within the corresponding XAML file. As shown in figure 1.7 through the `Loaded` attribute, we can actually specify the name of the event handling method within our XAML code. When using this approach, the compiler expects a method named after the value of the attribute within the code-behind file. It is our responsibility to ensure that the method accepts the correct number, and type of parameters.

In addition to handling events, the code-behind file may also be used to define public properties which can then be set through our XAML code. This practice is most often used when creating custom controls, which is discussed in Chapter 12. While adding properties to classes is a common task, it is probably more common to add classes to namespaces.

### 1.3.2 Namespaces

A namespace provides a way of organizing related objects within a common grouping. These groupings, or namespaces, give us a way to define where the compiler should look for a type. In order to specify where to look, we must reference a namespace within the root element of a XAML file. The following XAML file illustrates the usage of two namespaces.

example 1.1 A basic XAML file referencing two namespaces.

```
XAML <Canvas x:Name="myCanvas"
L     xmlns="http://schemas.microsoft.com/client/2007"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     Loaded="Page_Loaded"
     x:Class="Namespace01.Page;assembly=ClientBin/namespace01.dll"
     Width="640" Height="480"
     Background="White">
     <TextBlock x:Name="myTextBlock" Text="Hello"
         FontFamily="Verdana" FontSize="12"
         MouseEnter="TextBlock_MouseHover" />
</Canvas>
```

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=369>

As example 1.1 illustrates, we are permitted to summon multiple namespaces within a single XAML file. When we are referencing multiple namespaces, it is important to acknowledge that each namespace must be uniquely prefaced. For instance, the "x" prefix in example 1.1 is used in association with the <http://schemas.microsoft.com/winfx/2006/xaml> namespace. While at the same time, the <http://schemas.microsoft.com/client/2007> namespace does not use a prefix.

These two namespaces that we just mentioned will be used in almost every Silverlight application you work with or see. These namespaces are generally defined in the following manner and expose these features to your Silverlight applications:

`xmlns="http://schemas.microsoft.com/client/2007"` – This namespace provides our applications with the core Silverlight elements. For this reason, this namespace generally omits a prefix, thus making it the default namespace within the page. This enables us to reference elements within this specific namespace without having to include the prefix.

`xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"` - This namespace provides functionality that is common across XAML. It is important to remember that XAML is utilized by other technologies such as WPF and Windows Workflow Foundation (WF), all of which need access to common features such as `Name`, `Key`, and `Class` properties.

In addition to these two commonly used namespaces; Silverlight gives us the flexibility to reference other namespaces including those within our own custom assemblies. When another assembly is referenced, it gets copied into a subdirectory of your Silverlight application called "ClientBin." In fact, when you compile your Silverlight application, it itself gets compiled into an assembly that gets placed in this directory. We will discuss the application model itself a little bit later, for now though, in order to reference these assemblies, we simply need to define a new namespace, which includes a prefix, namespace, and assembly. The following example illustrates this concept:

example 1.2 This example references a custom assembly called "MyAssembly.dll" that contains a namespace called "MyNamespace". In order to access the elements within "MyNamespace", we have defined the "my" prefix.

```
XAML <Canvas x:Name="parentCanvas"
L     xmlns="http://schemas.microsoft.com/client/2007"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     xmlns:my="clr-namespace:MyNamespace; [CA]
assembly=ClientBin/MyAssembly.dll"
     Loaded="Page_Loaded"
     x:Class="XAML02.Page;assembly=ClientBin/XAML02.dll"
     Width="640" Height="480"
     Background="White">
     <my:MyElement x:Name="myElement1" />
</Canvas>
```

As example 1.2 illustrates, referencing other elements, including custom elements, only requires us to provide a couple of more details. While the process of creating custom

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=369>

elements is detailed in Chapter 12, example 1.2 also contains a few other XAML related concepts that we should discuss. The least of which, is the incredibly powerful compound property feature.

### 1.3.3 Compound Properties

While the illustration in figure 1.6 shows the relationship between XAML and .NET types and properties, there was one detail of properties that was omitted. Properties within XAML may consist of elements significantly more complex and detailed than primitive types. For instance, the following XAML uses another kind of .NET object, called a `LinearGradientBrush`, to define the background of a `Canvas`.

example 1.3 A Canvas with a gradient background.

```
XAML <Canvas x:Name="myCanvas"
L     xmlns="http://schemas.microsoft.com/client/2007"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     Width="640" Height="480">
     <Canvas.Background>
     <LinearGradientBrush>
         <GradientStop Offset="0" Color="White"/>
         <GradientStop Offset=".5" Color="Navy"/>
         <GradientStop Offset="1" Color="White"/>
     </LinearGradientBrush>
     </Canvas.Background>
</Canvas>
```

The concept of a `GradientBrush` is discussed in Chapter 8, however, as you can see, we defined the `Background` element inside of the XAML hierarchy. This approach is enabled by the concept of a "compound property."

A compound property enables us to use the syntax of `TypeName.PropertyName` within an element, to define more complex items within an element. This powerful feature gives us the ability to easily see the hierarchy of an object structure. In addition, it gives us a significant amount of flexibility when we are creating control templates, which we will discuss in Chapter 10. But first, there is another kind of property that is equally important.

### 1.3.4 Attached Properties

An attached property is actually a property that is specified within another element, other than the element that references it. While this seems like a mouthful, it's really not. You can easily recognize an attached property by its consistent syntax of `AttachedElement.PropertyName`. These special attributes are generally used within the context of layout panels, which are discussed in Chapter 3.

However, as an example, let's pretend we need to define a `Rectangle` within a `Canvas`. This `Rectangle` will have an offset of 10 pixels from the top and 10 pixels from the left of the `Canvas`. As the two previous sentences illustrate, the subject (`Rectangle`)

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=369>

is presented relative to the parent (`Canvas`). Thus, as the following example shows, the subject is attached to the parent.

example 1.4 An example showing the use of two attached properties in action.

```
XAM <Canvas x:Name="parentCanvas"
L   xmlns="http://schemas.microsoft.com/client/2007"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Width="640" Height="480" Background="White">
   <Rectangle Canvas.Left="10" Canvas.Top="10"
   Height="40" Width="120" Fill="Silver" />
</Canvas>
```

Example 1.4 uses the `Canvas.Left` and `Canvas.Top` attached properties to position the `Rectangle` within the `Canvas`. As we can clearly see, the attached properties are set just like traditional properties.

## ***1.4 Blending it Together***

Now that we have a fundamental understanding of the value that Silverlight brings to the web, we can begin learning about the application development experience. To begin our journey we will create a basic media player using Silverlight.

This walkthrough is designed to illustrate the seamless integration between the Microsoft Visual Studio and Blend tools. Though the tasks may seem tedious at first, it is important to have an understanding of what each tool is geared for. When we are done with the exercises in this section, we will have a media player that looks like the following:

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=369>

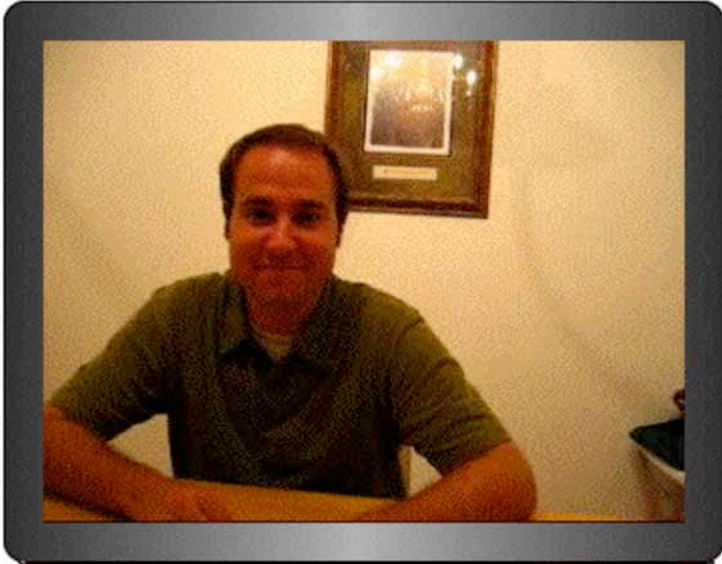


figure 1.8 The finished product of this "Blending it Together" session.

walkthrough 1.1 Creating a Silverlight media experience with Visual Studio.

1. Launch "Microsoft Visual Studio 2008."
2. Select File -> New -> Project...
3. From the "New Project" dialog, select the "Silverlight" project type. Then select the "Silverlight Project" template. (NOTE: If you do not have the Visual Studio Extensions for Silverlight installed, you will not have these options. Please install the Visual Studio Extensions for Silverlight to continue.)
4. Once you have selected the "Silverlight Project" template, enter the following project properties within the "New Project" dialog:

Name:	Welcome
Location:	C:\Silverlight in Action\ <input type="button" value="Browse..."/>
Solution Name:	Welcome <input checked="" type="checkbox"/> Create directory for solution

5. Click OK. Once the Silverlight project is loaded within Visual Studio, The Page.xaml file will be opened.
6. Within the Page.xaml file, there will be a **Canvas** element. Within the **Canvas** element, Add the following XAML:

```
<MediaElement x:Name="myMediaElement"
  Canvas.Left="20" Canvas.Top="20"
  Source="welcome.wmv[CAC1]" />
```

This code segment uses a **MediaElement** object to reference a video file over the internet. Once you have inserted the **MediaElement**, the Page.xaml file should look similar to the following:

Please post comments or corrections to the Author online forum at <http://www.manning-sandbox.com/forum.jspa?forumID=369>

```
<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="Welcome.Page;assembly=ClientBin/Welcome.dll"
  Width="640"
  Height="480"
  Background="White"
>
  <MediaElement x:Name="myMediaElement"
    Canvas.Left="20" Canvas.Top="20"
    Source="welcome.wmv[CAC2]" />
</Canvas>
```

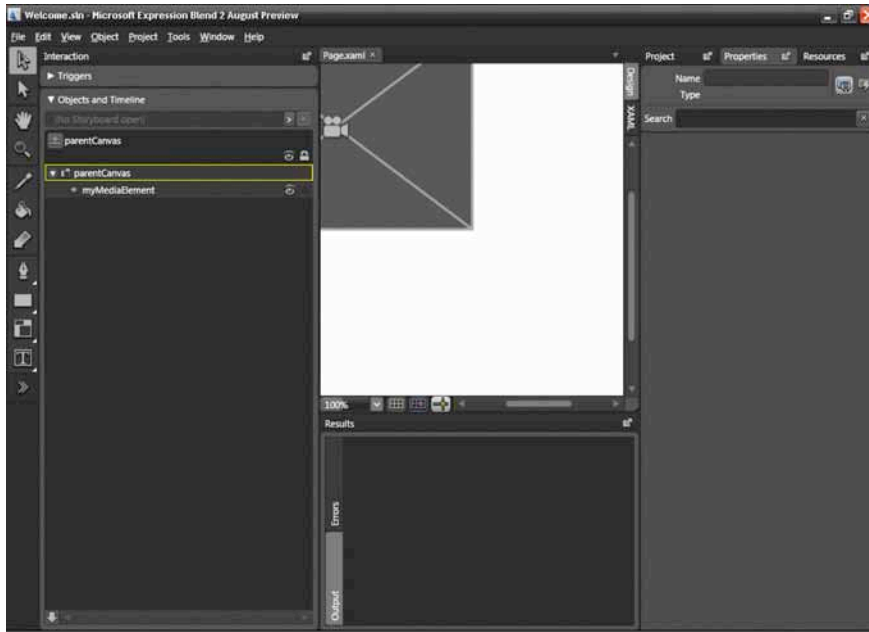
7. Replace the "welcome.wmv" file with a video of your choice.
8. Press F5 to run the application.
9. After watching the video, close the web browser window that was launched.
10. Leave Microsoft Visual Studio Open!

In walkthrough 1.1 we successfully created a Silverlight application that plays media related content by adding one simple element. As engaging as the application is, we will now turn our focus to using Microsoft Expression Blend to stylize the media.

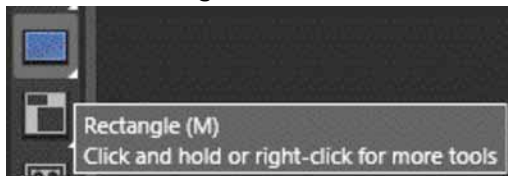
walkthrough 1.2 Matting a MediaElement using Blend.

1. Launch "Microsoft Expression Blend 2"
2. Select "Open Project..." from the startup dialog, or "File -> Open Project..." depending upon your preference.
3. Browse to the directory where you created the project in walkthrough 1.1.
4. Select the "Welcome.sln" file and click the "Open" button. Alternatively, you could select the "Welcome.csproj" file because Blend has built in support for both the solution and project file schemas. Once the project is loaded, you should see something similar to the following:

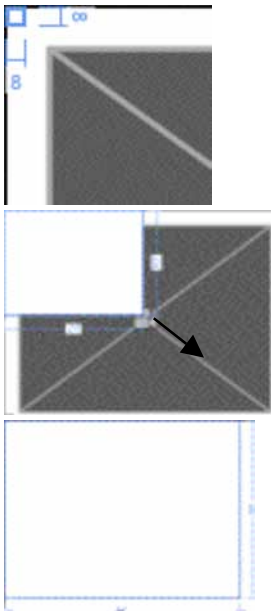
Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=369>



5. From the Toolbox along the left side of the Blend, select the Rectangle tool. The Rectangle tool looks like the following:



6. With the Rectangle tool selected, draw a **Rectangle** from the upper left corner of the **Canvas** to the lower right corner of the **MediaElement** that is already there.

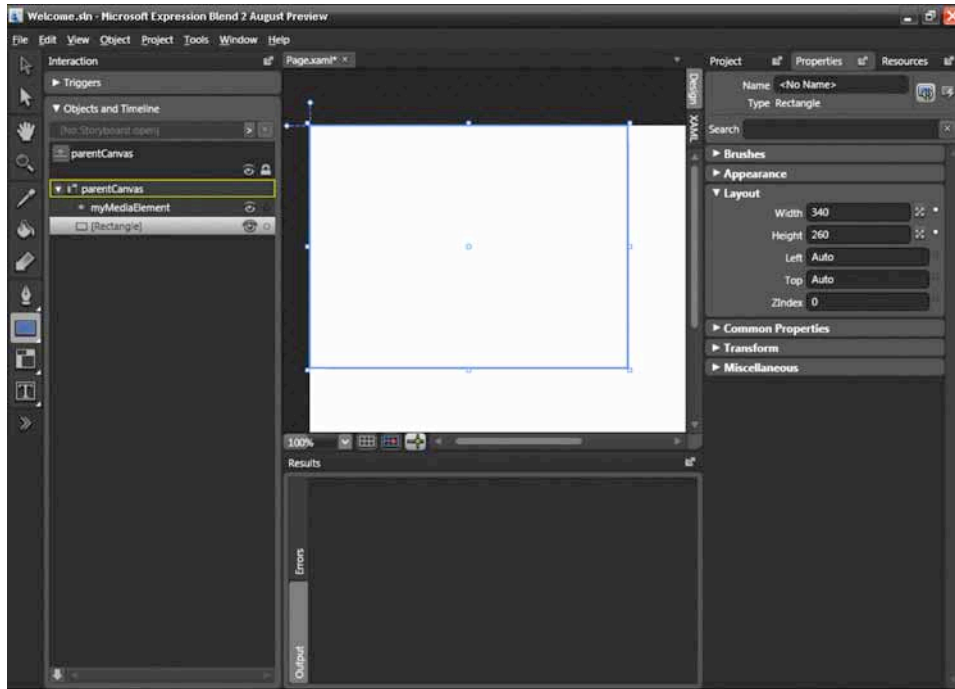


From the upper left corner, drag the mouse down and to the right.

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=369>

Continue to drag until the MediaElement is covered by the rectangle.  
When the element is covered, release the mouse.

- Next we will ensure that this **Rectangle** provides a nice mat for our **MediaElement** by setting the **Height** and **Width** properties. In order to this, please ensure that the “Properties” panel is selected within the upper right corner of Blend. If it is not selected, please select it.



- Manually set the **Width** and **Height** properties within the Layout category of the Properties panel. The **Width** should be set to “360” and the **Height** should be set to “280”.



- Select the “Brushes” category.
- Select the “Gradient brush” tab within the “Brushes” category and ensure that the “Fill” property is selected.

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=369>



11. We are going to fill our **Rectangle** with a gradient by doing the following:



Click the left-most gradient and set the value to a dark gray such as "#FF383737" as shown in the picture.



Add a new gradient stop by clicking the middle of the gradient slider. Set the value of this gradient stop to a light gray such as "#FFB4B4B4".

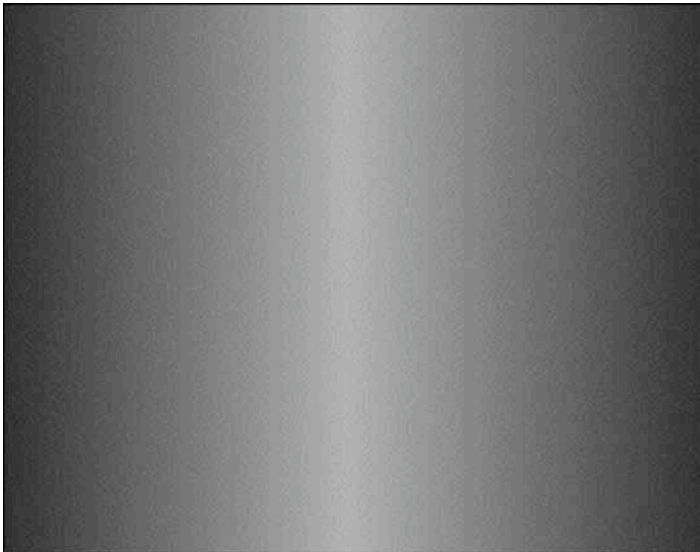
Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=369>



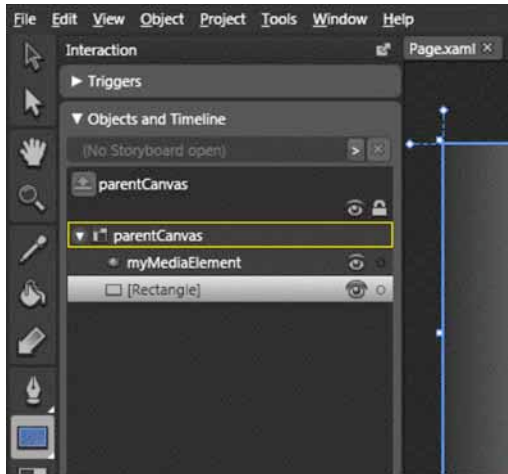
Set the third and right-most gradient stop to the same gray we used for the left-most gradient stop (a dark gray or "#FF383737")



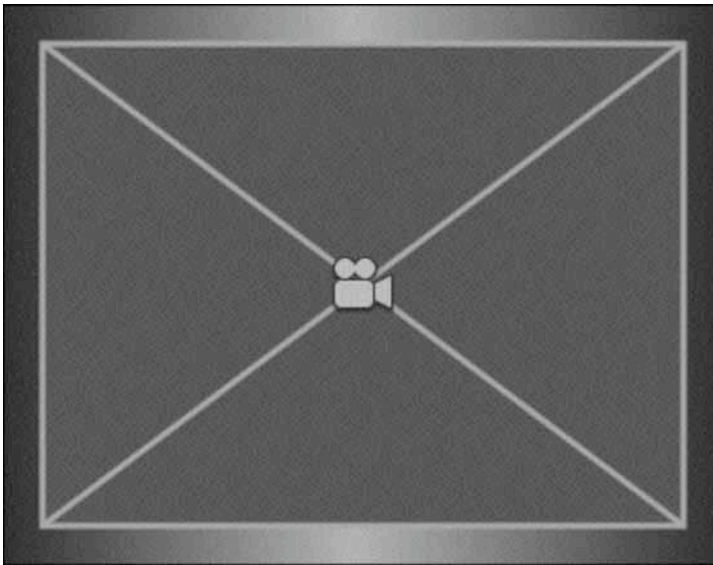
12. At this point, we have a **Rectangle** that is on top of the **MediaElement** and thus, hiding it. If you were to run the project, you would hear the **MediaElement** playing, but you would not be able to see it. Your project should look something like this:



13. Select the "[Rectangle]" within the "Objects and Timeline" category within the "Interaction" panel which is just to the right of the toolbox we used to draw the **Rectangle**.



14. We need to place the **Rectangle** behind the **MediaElement**. In order to do this, make sure the **Rectangle** is selected, as shown above, and then go to Object -> Order -> Send to Back.
15. Now the **MediaElement** should be on top of the **Rectangle** like this:



16. Press F5 to run the application. Notice how this is consistent with Visual Studio.
17. Close the web browser window that was launched.

Walkthrough 1.2 showed us how we can open Silverlight projects created from Visual Studio and stylize them within Blend. In addition, we were also introduced to some of the basic elements available within Blend. As you can see, Blend makes it easy to reach beyond traditional web graphics and add richer elements such as gradients. As the next section shows, these elements seamlessly transition back to our Visual Studio environment.

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=369>

walkthrough 1.3

1. Either Alt-Tab back to Microsoft Visual Studio or select it from the taskbar.
2. Notice: You will be prompted to Reload the Page.xaml file in a dialog that will look like the following:



3. Click "Yes" in this dialog box.
4. Notice: The XAML code is automatically updated to include the XAML for the Rectangle that we created in walkthrough 1.2.
5. Press F5 to run the application just to show how everything was migrated over without any migration process.
6. Close the web browser window that was launched.
7. Select View->Code to open the Page.xaml.cs file.
8. Within the `Page_Loaded` method body, add the following line of code:

```
myMediaElement.MouseLeftButtonUp += new  
    MouseEventHandler(myMediaElement_MouseLeftButtonUp);
```

After this line of code is added, the `Page_Loaded` method should look like the following:

```
public void Page_Loaded(object o, EventArgs e)  
{  
    // Required to initialize variables  
    InitializeComponent();  
  
    myMediaElement.MouseLeftButtonUp += new  
        MouseEventHandler(myMediaElement_MouseLeftButtonUp);  
}
```

9. Implement the following event handler:

```
void myMediaElement_MouseLeftButtonUp(object sender,  
    MouseEventArgs e)  
{  
    if (myMediaElement.CurrentState == "Playing")  
        myMediaElement.Pause();  
    else  
        myMediaElement.Play();  
}
```

10. Press F5 to run the application.

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=369>

At this point we have successfully created a basic media player that illustrates many of the core concepts of Silverlight. The details surrounding these concepts will be exposed as we progress through this book.

## ***1.5 Summary***

Silverlight is Microsoft's offering for delivering rich, cross-browser, cross-platform user experiences over the internet. These experiences have the potential of revolutionizing the internet and can be witnessed by approximately 96% of the internet's population.

With the power of the .NET Framework, Silverlight enables greater developer productivity, opportunity, and choice. In addition, the collaboration enabled with designers through XAML, and a consistent project schema between tools, encourages higher-quality products built with smaller amounts of effort. In order to deliver these high quality interactive products, we need to have an in-depth knowledge of how to actually enable a web page to host Silverlight content. This is the purpose of Chapter 2.