

1

Learning the Nuts and Bolts of Silverlight 4

In this chapter, we will cover the following topics:

- ▶ Getting our environment ready to start developing Silverlight applications
- ▶ Creating our first service-enabled and data-driven Silverlight 4 application using Visual Studio 2010
- ▶ Using the workflow between Visual Studio 2010 and Blend 4
- ▶ Using source control in Visual Studio 2010 and Blend 4
- ▶ Deploying a Silverlight application on the server

Introduction

While we assume some basic knowledge of Silverlight for this book, we also know that developers have very little time to grasp all the new technologies that keep coming out. Therefore, this first chapter contains all that we need to know to get going with Silverlight. We'll also guide you through the required tools and installations for a perfect Silverlight development environment.

Silverlight was released in the first half of 2007, and since then, it has created a lot of buzz. While ASP.NET is a server-side development platform, with the arrival of Silverlight, the focus has shifted to the **client side** again. A Silverlight application runs in **the browser** of the client and on a specific version of the **Common Language Runtime (CLR)**.

A big benefit for developers is that Silverlight uses .NET from version 2 onwards. It has a trimmed-down version of the **Base Class Library (BCL)** that is impressively extended considering the size of the Silverlight plugin (less than 5 MB). Because of the similarities, many skills achieved from developing applications in the full .NET framework can be leveraged for the creation of Silverlight applications.

Silverlight itself can be considered as a trimmed-down version of its desktop counterpart, **Windows Presentation Foundation (WPF)**. Between Silverlight 4 and WPF 4, there are still some differences. Some features are included in WPF 4, but aren't in Silverlight 4 and vice versa. It's possible to reuse code written for one technology in the other. However, upfront planning is required to ensure a smooth transition between the two technologies. Microsoft has released a whitepaper based on this aspect that provides more information on how to write applications that target both Silverlight and WPF. This document can be found at <http://wpfslguidance.codeplex.com>.

With the release of Silverlight 2, Microsoft made it clear that Silverlight is aimed at both creating rich and interactive applications, and next-level enterprise applications in the browser. The latter can be easily seen with the addition of a rich control set, support for many types of services and platform features such as data binding.

Due to its client-side characteristics, Silverlight applications need to perform particular tasks to get data. It doesn't support client-side databases—not even in version 4. The way to retrieve data is through services. Silverlight 3 brought some interesting features to the platform in this area such as support for binary XML, the WCF RIA services, and simplified duplex service communication. Silverlight 4 continued in the same manner, with improvements in data binding, support for `net.tcp` communication, cross-domain access to services by means of Trusted Silverlight applications, and much more. All these added features are a proof of the commitment Microsoft is making to position Silverlight as a platform for building enterprise applications.

In this chapter, we'll get you up and running with Silverlight. While this book is aimed at developers who already have a basic knowledge of Silverlight, this chapter can act as a refresher. We'll also look at getting your environment correctly set up so that you enjoy developing Silverlight applications.

Getting our environment ready to start developing Silverlight applications

In this recipe, we'll look at what we need to install to start developing Silverlight applications. We'll learn about the basic tools that we need as a developer and also take a look at the designer tools that can come in handy for developers as well.

How to do it...

To start developing Silverlight applications, we'll need to install the necessary tools and SDKs. Carry out the following steps in order to get started:

1. We need to make sure we install Visual Web Developer Express 2010 (available for free at <http://www.microsoft.com/express/downloads/>) or Visual Studio 2010 (trial version available at <http://www.microsoft.com/visualstudio/en-us/download>).
2. Go to <http://www.silverlight.net/getstarted/> to download and install the Silverlight 4 Tools for Visual Studio 2010. Open Visual Studio 2010 after installation. Visual Studio 2010 ships with Silverlight 3 templates installed out of the box, the tools add support for version 4.
3. Go to <http://www.microsoft.com/expression/> to download and install Blend 4.

How it works...

For Silverlight development, the minimum that we need are the developer tools. These will integrate with Visual Studio 2008 (if you're using Silverlight 3) or 2010. In Visual Studio 2010, a nice, visual designer is added for editing our XAML code. In the 2008 version, this designer doesn't exist. When installing the developer tools for Silverlight 4, the following components are automatically downloaded and installed:

- ▶ Silverlight 4 developer runtime
- ▶ Silverlight 4 software development kit and Visual Studio project support
- ▶ WCF RIA services

We can write XAML code using Visual Studio. However, if you're serious about designing, you might want to consider using Microsoft Expression Blend. This tool, primarily aimed at designers, should be seen as an application that generates XAML for us by means of a rich number of options and an easy-to-use interface. It also integrates nicely with Visual Studio and source control software integration is available as well.

See also

After having installed all the necessary tools, it might be worth taking a look at the *Creating our first service-enabled and data-driven Silverlight 4 application using Visual Studio 2010* recipe as well as the *Using the workflow between Visual Studio 2010 and Blend 4* recipe. In these recipes, we create an entire application in Visual Studio 2010 and Blend 4 respectively.

Creating our first service-enabled and data-driven Silverlight 4 application using Visual Studio 2010

In this recipe, we'll build a very simple Silverlight application that uses techniques that are explained in much more detail later on in the book. We'll be using **data binding**, which is a technique to easily connect data to the **user interface (UI)** and connect to a **Windows Communication Foundation (WCF)** service.

However, the main goal is to get a grip on the basics of Silverlight by trying to answer questions such as how a Silverlight application is built, what the project structure looks like, what files are created, and what is their use.

Getting ready

To get started with Silverlight application development, make sure that you have installed Visual Studio along with the necessary tools and libraries as outlined in the previous recipe.

We are building this application from the ground up. However, the finished product can be found in the `Chapter01/SilverlightHotelBrowser` folder in the code bundle that is available on the Packt website.

How to do it...

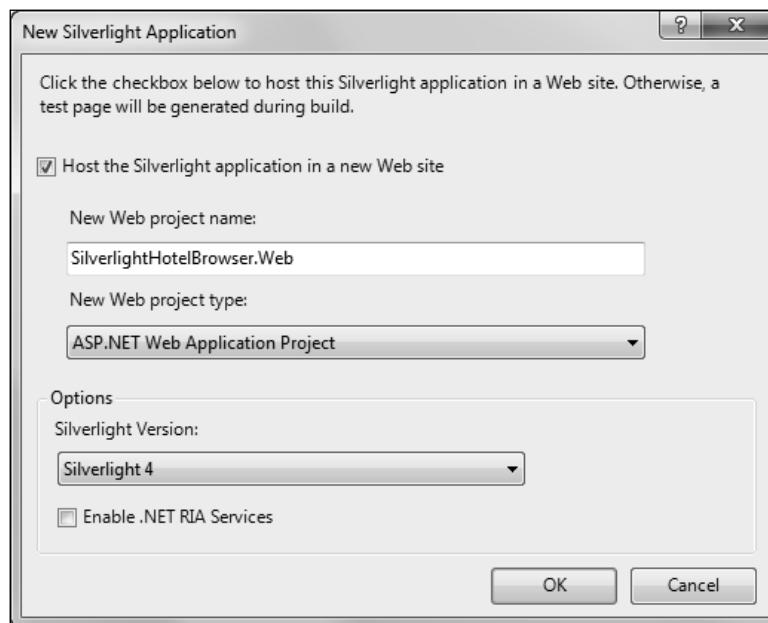
Our first Silverlight application allows the user to view the details of a hotel that is selected in a `ComboBox` control. The hotel information is retrieved over a service and is used for filling the `ComboBox` and the details shown in several `TextBlock` controls that are placed in a grid.

The following screenshot shows the interface of the application:

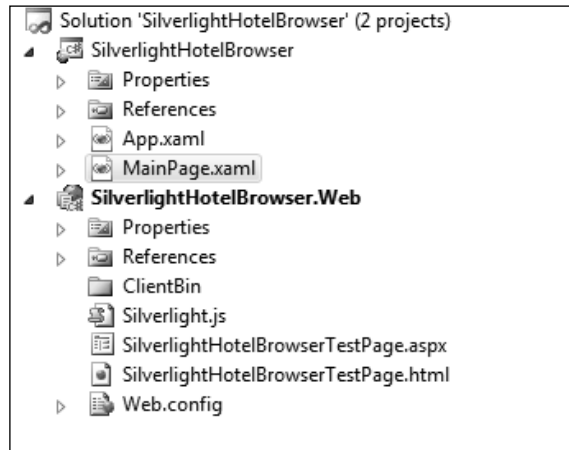


To start building any Silverlight application, we'll need to perform the following steps:

1. Open Visual Studio 2010 with the Silverlight 4 tools installed. Once inside the **Integrated Development Environment (IDE)**, go to **File | New | Project...** In the **New Project** dialog that appears, select the **Silverlight** node under **Visual C#** and select **Silverlight Application**. Name the application as **SilverlightHotelBrowser** and click the **OK** button. In the dialog that appears as shown in the next screenshot, select **ASP.NET Web Application Project** as the type of web project that will be used to host the Silverlight application. Also, make sure that **Silverlight 4** is selected as the target version of Silverlight 4.



2. After Visual Studio has finished executing the project template, two projects are created in the solution: the Silverlight project and a web application project that is responsible for hosting the Silverlight content. The created solution structure can be seen in the following screenshot:



3. Our service will return the hotel information. A hotel can be represented by an instance of the `Hotel` class. This class should be included in the web project—`SilverlightHotelBrowser.Web`.

There are a few things to note about this class:

- This class has a `DataContract` attribute attached to it. This attribute is required to specify that this class can be serialized when sent over the wire to the client application.
- Each property is attributed with the `DataMember` attribute. When adding this attribute to a property, we specify that this property is a part of the contract and that it should be included in the serialized response that will be sent to the client.

The following code defines this class:

```
[DataContract]
public class Hotel
{
    [DataMember]
    public string Name { get; set; }
    [DataMember]
    public string Location { get; set; }
    [DataMember]
    public string Country { get; set; }
}
```

```
[DataMember]
public double Price { get; set; }
}
```

4. We'll now add a WCF service to the web project as well. Right-click on `SilverlightHotelBrowser.Web` and select **Add | New Item....** Add a **Silverlight-enabled WCF Service** by selecting the **Silverlight** node under **Visual C#** and name it as **HotelService**. Click the **Add** button. Two files are added, namely, `HotelService.svc` and `HotelService.svc.cs`.
5. In this service class, we can now add a method that returns a hardcoded list of hotels. Remove the `DoWork` method and replace it with the following code:

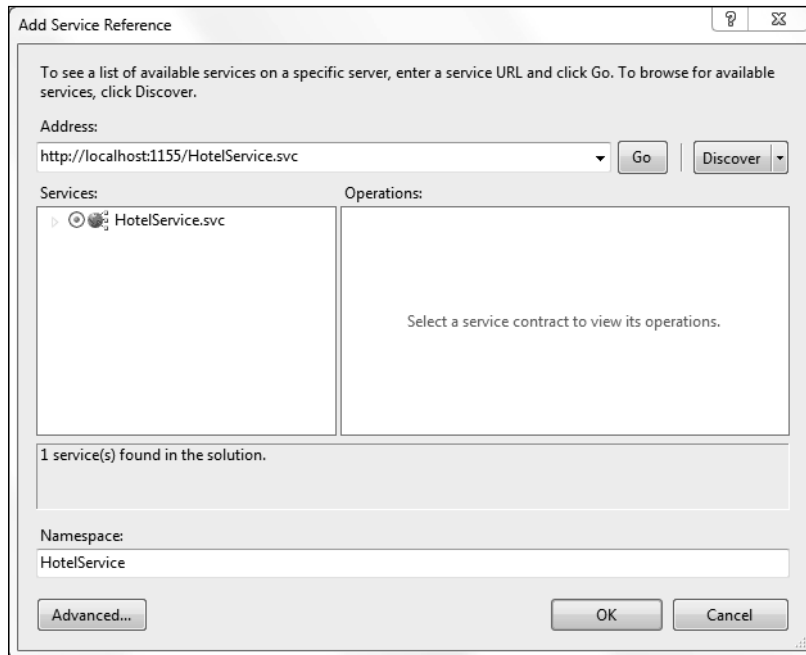
```
[ServiceContract(Namespace = "")]
[AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
public class HotelService
{
    [OperationContract]
    public List<Hotel> GetHotels()
    {
        return new List<Hotel>
        {
            new Hotel
            {
                Name = "Brussels Hotel",
                Price = 100,
                Location = "Brussels",
                Country = "Belgium"
            },
            new Hotel
            {
                Name = "London Hotel",
                Price = 200,
                Location = "London",
                Country = "United Kingdom"
            },
            new Hotel
            {
                Name = "Paris Hotel",
                Price = 150,
                Location = "Paris",
                Country = "France"
            },
            new Hotel
            {
```

```
        Name = "New York Hotel",  
        Price = 230,  
        Location = "New York",  
        Country = "USA"  
    }  
};  
}
```

Note the attributes that have been used in this class. The `ServiceContract` attribute specifies that the class contains a service contract that defines what functionality the service exposes. The `OperationContract` attribute is added to operations which can be invoked by clients on the service. This effectively means that if you add methods to the service without this attribute, it can't be invoked from a client

6. Now we'll build the solution and if no errors are encountered, we're ready for takeoff—takeoff for writing Silverlight code.

Let's first make the service known to the Silverlight application. Right-click on the Silverlight application and select **Add Service Reference...** In the dialog box that appears, as shown in the following screenshot, click on **Discover** and select your service. As it is in the same solution, the service will appear. Enter **HotelService** in the **Namespace** field.



Click on the OK button to confirm. The service is now usable from the Silverlight application.

7. The UI was shown earlier and is quite easy. The XAML code for the Grid named LayoutRoot inside the MainPage.xaml file is as follows:

```
<Grid x:Name="LayoutRoot" Width="400" Height="300"
      Background="LightGray">
  <Grid.RowDefinitions>
    <RowDefinition Height="50"></RowDefinition>
    <RowDefinition></RowDefinition>
  </Grid.RowDefinitions>
  <ComboBox x:Name="HotelComboBox" Width="250"
            SelectionChanged="HotelComboBox_SelectionChanged"
            DisplayMemberPath="Name"
            VerticalAlignment="Center">
</ComboBox>
  <Grid x:Name="HotelDetailGrid" Grid.Row="1"
        VerticalAlignment="Top">
    <Grid.RowDefinitions>
      <RowDefinition></RowDefinition>
      <RowDefinition></RowDefinition>
      <RowDefinition></RowDefinition>
      <RowDefinition></RowDefinition>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition></ColumnDefinition>
      <ColumnDefinition></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <TextBlock x:Name="NameTextBlock"
              Grid.Row="0"
              Grid.Column="0"
              FontWeight="Bold"
              Text="Name: "
              HorizontalAlignment="Right">
</TextBlock>
    <TextBlock x:Name="NameValueTextBlock"
              Grid.Row="0"
              Grid.Column="1"
              Text="{Binding Name}">
</TextBlock>
    <TextBlock x:Name="LocationTextBlock"
              Grid.Row="1"
              Grid.Column="0"
              FontWeight="Bold"
              Text="Location: "
              HorizontalAlignment="Right">
```

```
</TextBlock>
<TextBlock x:Name="LocationValueTextBlock"
           Grid.Row="1"
           Grid.Column="1"
           Text="{Binding Location}">
</TextBlock>
<TextBlock x:Name="CountryTextBlock"
           Grid.Row="2"
           Grid.Column="0"
           FontWeight="Bold"
           Text="Country: "
           HorizontalAlignment="Right">
</TextBlock>
<TextBlock x:Name="CountryValueTextBlock"
           Grid.Row="2"
           Grid.Column="1"
           Text="{Binding Country}">
</TextBlock>
<TextBlock x:Name="PriceTextBlock"
           Grid.Row="3"
           Grid.Column="0"
           FontWeight="Bold"
           Text="Price: "
           HorizontalAlignment="Right">
</TextBlock>
<TextBlock x:Name="PriceValueTextBlock"
           Grid.Row="3"
           Grid.Column="1"
           Text="{Binding Price}">
</TextBlock>
</Grid>
</Grid>
```

8. In the code-behind class, `MainPage.xaml.cs`, we connect to the service and the results are used in a data binding scenario. We won't focus here on what's actually happening with the service call and the data binding; all this is covered in detail later in this book.

```
public MainPage()
{
    InitializeComponent();
    HotelService.HotelServiceClient proxy = new
        SilverlightHotelBrowser.HotelService.HotelServiceClient();
    proxy.GetHotelsCompleted += new EventHandler
        <SilverlightHotelBrowser.HotelService.
        GetHotelsCompletedEventArgs>
        (proxy_GetHotelsCompleted);
}
```

```

        proxy.GetHotelsAsync();
    }
    void proxy_GetHotelsCompleted(object sender,
        SilverlightHotelBrowser.HotelService.
        GetHotelsCompletedEventArgs e)
    {
        HotelComboBox.ItemsSource = e.Result;
    }

    private void HotelComboBox_SelectionChanged(object sender,
        SelectionChangedEventArgs e)
    {
        HotelDetailGrid.DataContext = (sender as ComboBox)
            .SelectedItem as HotelService.Hotel;
    }

```

8. We will compile the solution again and then press the *F5* button. The application should now run, allowing us to select a hotel in the `ComboBox`. Each selection change will trigger an event that shows the details of the selected item using data binding.

How it works...

Silverlight applications always have to run in the context of the browser. That's the reason why Visual Studio prompts us initially by asking how we want to host the Silverlight content.

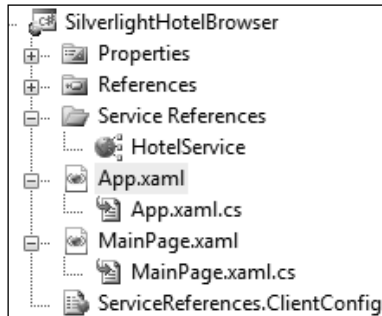


Note that Silverlight 3 added out-of-browser capabilities to its previous version, thereby allowing Silverlight applications to run "standalone". However, they still run inside the browser sandbox. Silverlight 4 added the option to run a Silverlight application out of browser with elevated permissions, thereby giving it more permissions on the local system. We'll be looking at these later on in this book.

The default option is the **ASP.NET Web Application Project**. This option gives us the maximum number of possibilities for the configuration of the host project. It's the option that we will be using the most throughout this book because of the configuration options it offers towards working with services. The second option is **ASP.NET Web Site** and is a file-based website known from ASP.NET 2.0. Finally, we can also uncheck the **Host the Silverlight application in a new Web site** checkbox. This will result in Visual Studio generating an empty HTML page containing the Silverlight application whenever we build our solution. This option is not well suited for building Silverlight applications that work with services as we have no control over the generation process.

The solution and project structure

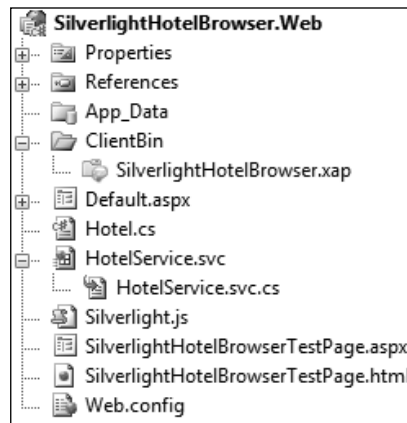
A new Silverlight solution thus contains normally two projects—a Silverlight project and a hosting application. Let's first take a look at the Silverlight project:



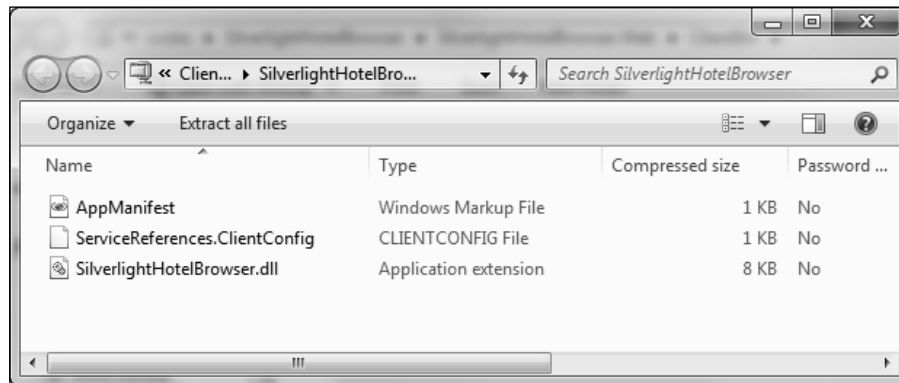
Silverlight applications contain XAML and C# (or VB.NET) files, among others. The XAML files contain the UI and are linked at runtime to the partial classes that make up the code-behind. By default, one page is added for free—called `MainPage`. It's not really a page, but a user control that is hosted. We can add UI code (such as controls, grids, and so on) to this file. We add UI logic in the code-behind.

One special case is the `App.xaml` file. It's the entry point of an application and is responsible for loading an instance of the `MainPage`, executing logic when an error occurs, and so on. Also, it can contain global resources such as styles that should be available over the entire application.

While building the solution, the Silverlight project is compiled into an assembly. In turn, this assembly—along with a manifest file that contains general information about the application and possible other resources—are wrapped into an XAP file. This XAP file is then copied into the hosting application. It shows up under the `ClientBin` directory in the web project as shown in the following screenshot:



The XAP file is basically a ZIP (archive) file. When renaming the `SilverlightHotelBrowser.xap` file to `SilverlightHotelBrowser.zip`, we can see the original files (manifest and assembly). The following screenshot shows the contents of the ZIP file:



The generated ASPX page as well as the HTML page refer to the XAP file located in the `ClientBin` directory.

Services

Data is not readily available to a Silverlight application on the client side. So we need to retrieve it from the server. In Silverlight, this is done using services. Services need to be accessed asynchronously in Silverlight, hence the declaration of the callback method—`proxy_GetHotelsCompleted`. Silverlight has many options to communicate with services. These are covered in the recipes of this book.

Data binding

We use the rich data binding features available in Silverlight to connect the data with the UI in this application. Data binding allows us to bind properties of objects (the data) to properties of controls. In this particular example, we bind a list of `Hotel` instances to the `ComboBox` using the `ItemsSource` property. While changing the selection in the control, the `HotelComboBox_SelectionChanged` event handler fires and the selected item—a `Hotel` instance—is set as the `DataContext` for the `HotelDetailGrid`. This grid contains the controls in which we want to show the details. Each of these controls uses a `Binding` markup extension in XAML to specify which property needs to be bounded.

See also

Data binding was used in this application. It's also the topic of Chapter 3, where we delve deep into what data binding has to offer. We also connected with a WCF service. Connecting and communicating with services is covered in Chapter 6 and Chapter 7.

Using the workflow between Visual Studio 2010 and Blend 4

Expression Blend (currently at version 4) is part of Microsoft's Expression Suite. It's a designer tool that allows designers to create compelling user experiences for use with WPF and Silverlight. While aimed at designers, it's a tool that should be in a Silverlight developer's toolbox as well. In some areas, it offers a richer designer experience than Visual Studio does. One of the best examples of this is the timeline that makes it easy to create time-based animations.

In this recipe, we'll look at how Visual Studio and Blend integrate. When used together, they help us create our applications faster. In the next chapter, we'll take another look at Blend—namely at its features that support data binding.

Getting ready

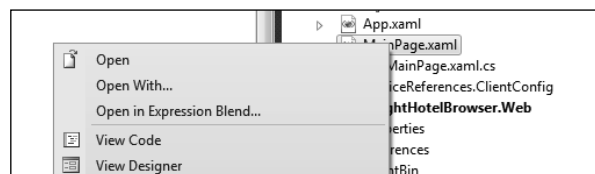
After having read the *Getting our environment ready to start developing Silverlight applications* recipe, you should have Expression Blend 4 installed.

In this recipe, we are creating the sample from scratch. The completed solution can be found in the `Chapter01/Blend` folder in the code bundle that is available on the Packt website.

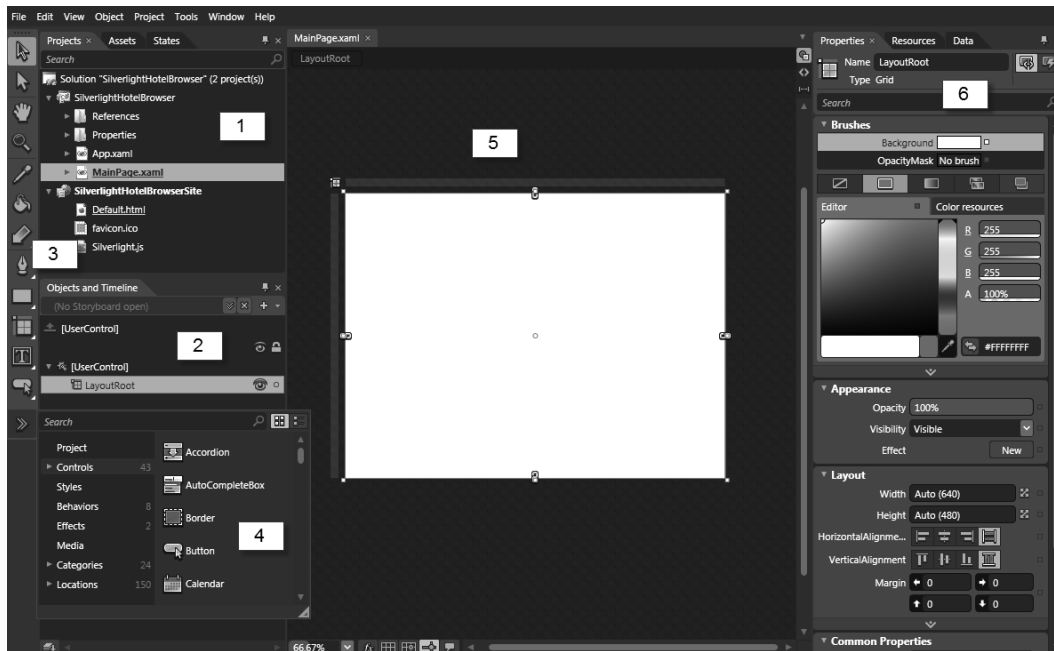
How to do it...

In this recipe, we'll recreate the Hotel Browser application. However, we'll do most of the work from Blend and switch back and forth to Visual Studio when it is recommended. We'll need to carry out the following steps:

1. Although we can start a new solution from Blend, we'll let Visual Studio create the solution for us. The main reason is that we'll be using services later on in this sample and working with services is easier if the hosting site is an ASP.NET web application. Adding an ASP.NET web application is possible from Visual Studio, but not from Blend. Therefore, open **Visual Studio 2010** and create a new Silverlight solution. Name it `SilverlightHotelBrowser` and make sure to select **ASP.NET Web Application Project** as the hosting website.
2. With the solution created in Visual Studio, right-click on one of the XAML files in the Silverlight project. In the context menu, select **Open in Expression Blend...** as shown in the following screenshot:



- Expression Blend will open up and its workspace will be shown. The following is a screenshot of the interface containing some named references:

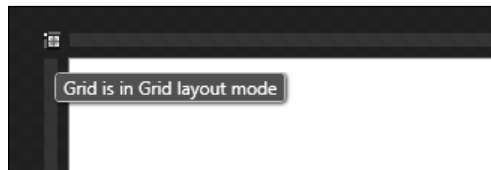


The following table describes some of the most important items on the Blend workspace:

Item	Name	Description
1	Projects window	Gives an overview of the loaded solution and its projects. It is comparable to the Solution Explorer in Visual Studio.
2	Objects and Timeline	By default, this window gives an overview of all the XAML objects in the currently loaded document. When we want to perform any action on an item (such as giving it a background color), we select it in the Objects and Timeline window. This opens the properties window for that item.
3	Toolbox	Comparable to what we know from Visual Studio, the toolbox contains all the tools available. Since Blend is a design tool, tools such as a Pen, Paint Bucket, and so on are available in the toolbox.
4	Assets window	The Assets window contains all controls (assets) that we can drag onto the design surface such as Buttons, ComboBoxes, and so on.

Item	Name	Description
5	Design workspace	This is where all the action takes place! We can drag items from the Toolbox or the Assets window, rearrange them and so on to create a compelling user experience.
6	Properties window	The Properties window allows us to change the properties of the selected item. We can change the color, layout properties, transform properties and so on.

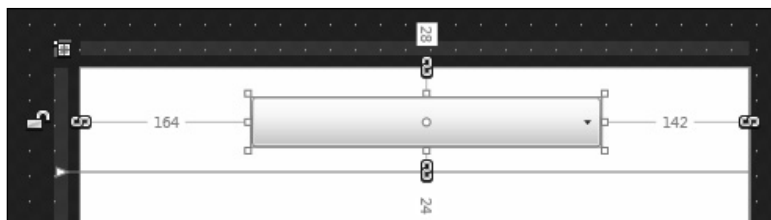
- Now that we know our way around the designer, we can get creative. We'll start with `MainPage.xaml` and split the main **Grid** (`Layout.Root`) into two rows. Instead of writing the XAML code for this, we'll do this in the designer. Click on the icon on the top left of the user control in the designer so that the Grid will be in the Grid layout mode. This can be seen in the following screenshot:



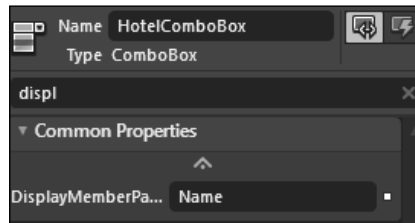
- Now, click on the left bar next to the user control to add a row. It's possible to change the height of the created row by dragging the handle. The following screenshot shows a row added to the Grid::



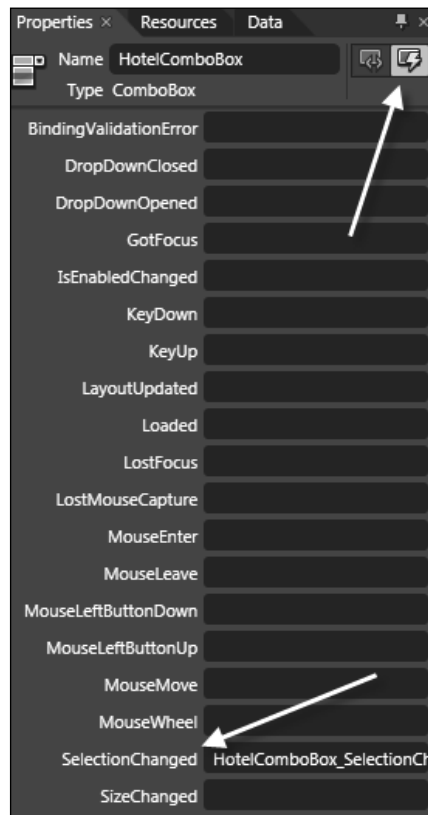
- Select the **ComboBox** in the **Assets** window. Use the search function in this window to find it more quickly. On the designer, drag to create an instance of the `ComboBox` and place it on the top row that was just created. This can be seen in the next screenshot:



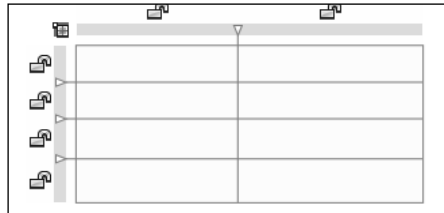
- In the **Properties** window, give this `ComboBox` the name **HotelComboBox** and set the **DisplayMemberPath** property to **Name**. In the following screenshot, note that we are making use of the **Search** functionality within the **Properties** window. Simply enter part of the name of the property you are looking for (here **displ**) and Blend will filter the available properties.



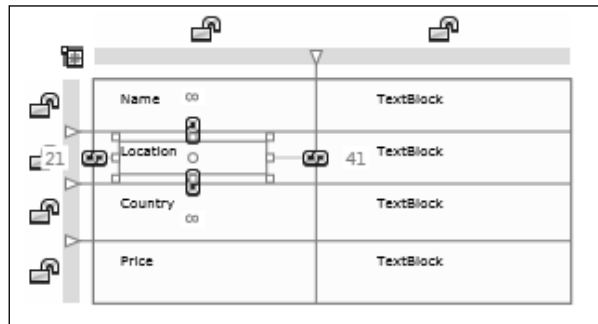
- With the `ComboBox` still selected in the **Properties** window, change to the **Events** view (top arrow in the next screenshot). In the list of events, double-click on the **SelectionChanged** event, so Blend will create an event handler (bottom arrow in the next screenshot).



- Let's now move back to the **Design** view of `MainPage.xaml`. Select the **Grid** item in the **Toolbox**. In the bottom cell of the `LayoutRoot` (the main grid control) drag to create a nested grid. Create four rows and two columns using the same technique as before. Columns are created quite logically by clicking on the top bar of the control. The result is shown in the following screenshot:



- With this **Grid** still selected, change the name to `HotelDetailGrid` in the **Properties** window.
- In each of the cells, drag a **TextBlock** from the **Toolbox**. For the **TextBlock** controls in the first column, change the `Text` property as shown in the following screenshot. Don't change the `Text` property of the controls in the second column; we'll look at these in the coming steps.



- Let's now change the background color of the **LayoutRoot** grid. To do this, select the **LayoutRoot** node in the **Objects and Timeline** window and in the **Properties** window, change the background by selecting a different color in the editor.
- In Chapter 2, we'll look at how we can make data binding in Blend easier. As of now, we'll just type the XAML code from Blend. In the top-right corner of the **Design Surface**, select either the **Split view** or the **XAML view**. Blend shows us the XAML code that it created in the background. Search for the `TextBlock` controls in the second column of the `HotelDetailGrid` and change it as shown in the following code. Note that the generated code might not always be exactly the same as values such as `Margin` could be different.

```

<TextBlock
  Margin="49,8,40,8"
  Grid.Column="1"
  Text="{Binding Name}"
  TextWrapping="Wrap"/>
<TextBlock
  Margin="49,8,40,8"
  Grid.Column="1"
  Grid.Row="1"
  Text="{Binding Location}"
  TextWrapping="Wrap"/>
<TextBlock
  Margin="49,8,40,8"
  Grid.Column="1"
  Grid.Row="2"
  Text="{Binding Country}"
  TextWrapping="Wrap"/>
<TextBlock
  Margin="49,8,40,8"
  Grid.Column="1"
  Grid.Row="3"
  Text="{Binding Price}"
  TextWrapping="Wrap"/>

```

14. The workflow between Blend and Visual Studio allows us to jump to Visual Studio for the tasks we can't achieve in Blend, for example, adding a WCF service and referencing it in the Silverlight project. In the **Projects** window, right-click on a file or a project and select **Edit in Visual Studio**. If Visual Studio is still open, it will reactivate. If not, a new instance will get launched with our solution.
15. In the website that was created with the project initialization (`SilverlightHotelBrowser.Web`), we need to have a service that will return the hotel information. A hotel is represented by an instance of the `Hotel` class as shown in the following code:

```

[DataContract]
public class Hotel
{
  [DataMember]
  public string Name { get; set; }
  [DataMember]
  public string Location { get; set; }
  [DataMember]
  public string Country { get; set; }
  [DataMember]
  public double Price { get; set; }
}

```

16. Of course, we need to add the service as well. To do this, add a **Silverlight-enabled WCF service** called `HotelService`. Replace the `DoWork` sample method with the following code:

```
[OperationContract]
public List<Hotel> GetHotels()
{
    return new List<Hotel>
    {
        new Hotel
        {
            Name = "Brussels Hotel",
            Price = 100,
            Location = "Brussels",
            Country = "Belgium"
        },
        new Hotel
        {
            Name = "London Hotel",
            Price = 200,
            Location = "London",
            Country = "United Kingdom"
        },
        new Hotel
        {
            Name = "Paris Hotel",
            Price = 150,
            Location = "Paris",
            Country = "France"
        },
        new Hotel
        {
            Name = "New York Hotel",
            Price = 230,
            Location = "New York",
            Country = "USA"
        }
    };
}
```

17. Perform a build of the project so that the service is built and is ready to be referenced by the Silverlight application.
18. In the Silverlight project, add a service reference by right-clicking on the project and selecting **Add Service Reference...** Click on the **Discover** button and the service should be found. Set the namespace to **HotelService**.

19. In the `MainPage.xaml.cs`, add the following code to load the hotels in the `ComboBox` control:

```
public MainPage()
{
    InitializeComponent();

    HotelService.HotelServiceClient proxy = new
        SilverlightHotelBrowser.HotelService.HotelServiceClient();
    proxy.GetHotelsCompleted += new
        EventHandler<SilverlightHotelBrowser.HotelService.
            GetHotelsCompletedEventArgs>(proxy_GetHotelsCompleted);
    proxy.GetHotelsAsync();
}

void proxy_GetHotelsCompleted(object sender,
    SilverlightHotelBrowser.HotelService.
    GetHotelsCompletedEventArgs e)
{
    HotelComboBox.ItemsSource = e.Result;
}
```

20. In the `SelectionChanged` event handler of the `ComboBox`, add the following code to load the details of a hotel once the user selects a different option:

```
private void HotelComboBox_SelectionChanged(object sender,
    System.Windows.Controls.SelectionChangedEventArgs e)
{
    HotelDetailGrid.DataContext = (sender as ComboBox).SelectedItem
        as HotelService.Hotel;
}
```

21. Build and run the application in Visual Studio.

With these steps completed, we have created the application using both Blend and Visual Studio. For an application as easy as this one, there is less profit in switching between the two environments. However, with larger applications requiring large teams containing both developers and designers, this strong integration can turn out to be very helpful.

How it works...

Visual Studio and Blend integrate nicely with each other. It's easy to jump from one application to the other. This allows a great workflow between designers and developers.

Designers can work in Blend and the changes made in this tool are automatically picked up by Visual Studio, and vice versa. This is achieved through the use of the same files (both code files and project files) by the two tools. A solution created in Blend will open in Visual Studio. The same holds true for a solution created in Visual Studio; Blend can work with it.

See also

In Chapter 2, we'll use some more features of Blend. We'll perform data binding directly from Blend.

Using source control in Visual Studio 2010 and Blend 4

When working on slightly larger projects in teams, source control is an absolute necessity. By far, the most popular source control system in the Microsoft world today is **Team Foundation Server (TFS)**. This recipe explains all that we need to get TFS to work with Silverlight applications in Visual Studio and Blend. It doesn't explain how to work with TFS itself.

Getting ready

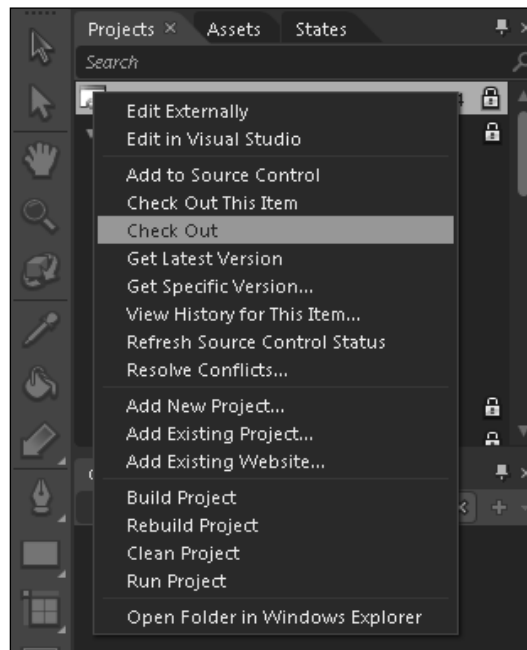
Before getting started, make sure that you have installed Blend 4 and the necessary developer tools as described in the *Getting our environment ready to start developing Silverlight applications* section.

How to do it...

To start using TFS as a versioning system with a Silverlight-enabled solution, we need to perform the following steps:

1. Using TFS source control with a Silverlight application in Visual Studio is exactly the same as using it with any other type of Visual Studio project. Team Explorer is used for this purpose and this is automatically installed out of the box with Visual Studio 2010. (For Visual Studio 2008, it has to be separately installed from <http://www.microsoft.com/downloads/details.aspx?FamilyID=0ed12659-3d41-4420-bbb0-a46e51bfca86&DisplayLang=en>.)
2. Blend 4 also supports source control out of the box. However, for Blend 3, TFS update KB967483 must be downloaded and installed from <http://code.msdn.microsoft.com/KB967483>. This update enables TFS source control for Expression Blend 3.

3. Once these are installed, we can use TFS source control from within Blend for any solution bound to the TFS source control. Right-click on any file in the Solution Explorer window to view the source control options just as we would do in Visual Studio's Solution Explorer window. We can now check out, check in, and merge files from Expression Blend.



How it works...

Team Foundation Server source control is the preferred way of enabling version control on our projects. Explaining in detail how to work with TFS is beyond the scope of this book, but the following are a few basic steps and references:

1. We must make sure that we have the correct permissions on our Team Foundation Server to handle the tasks we need to do. We need different permissions to (for example) create projects than to check out files. Have a look at this MSDN article to learn how to set these permissions:
<http://msdn.microsoft.com/en-us/library/ms252587.aspx>.
2. Next, we'll need to connect to TFS using Team Explorer. Have a look at this MSDN article to learn how to do that:
<http://msdn.microsoft.com/en-us/library/ms181474.aspx>.

3. We'll now need to create a workspace on our machine. We can look at the workspace as a local folder containing copies of the source-controlled files on the TFS. For more information, have a look at this MSDN article: <http://msdn.microsoft.com/en-us/library/ms181384.aspx>.
4. After we've created a local workspace, we can download the files from the TFS to that local folder. More information about this can be found at <http://msdn.microsoft.com/en-us/library/ms181385.aspx>.
5. We're now able to open our source-controlled solution in Blend and/or Visual Studio and check out files, merge files, add projects, and so on depending on the permissions.

There's more...

While working a lot with TFS, an interesting feature to download is the Team Foundation Server Power Tools. This is a set of extra features that is added to TFS and is mainly aimed at power users. It can be downloaded at <http://msdn.microsoft.com/en-us/teamsystem/bb980963.aspx>.

Commonly used terms in TFS

A complete glossary of TFS terms can be found at <http://msdn.microsoft.com/en-us/library/ms242882.aspx>. As a reference, the following are a few of the more commonly used ones along with their brief explanation:

Term	Explanation
TFS workspace	A TFS workspace is a location on the Team Foundation Server (TFS) where a record of changes between local files and corresponding repository files is stored. It can also be thought of as a copy of the client-side directory, a staging ground where local changes are persisted until they are checked into the server, or a collection of working folder mappings that can be viewed and edited.
Working folder	A working folder should be seen as a client-side representation of the TFS workspace. Binding the TFS workspace to the client-side working folder is done through a TFS workspace mapping.
Check in	Check in refers to the task of committing a pending change/pending changes to a TFS repository. When you check in pending changes, a new changeset is created on the server.
Check out	Check out refers to the task of notifying the TFS server that you are changing the status of a resource from locked to writeable. When you check out for edit, TFS appends an edit to that resource.
Get latest	Get latest refers to the task of retrieving the most recent version of a file from the TFS source control to your local working folder.

Deploying a Silverlight application on the server

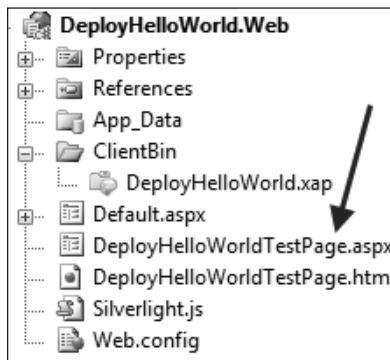
Once we have a Silverlight application ready, we will want to show it to the rest of the world. This means deploying it!

While Silverlight is a .NET technology, it doesn't require .NET to be installed on the server. Remember that it's a client-side technology. The Silverlight plugin on the client will download and run the application using the version of the **Common Language Runtime (CLR)** embedded in the Silverlight plugin. In this recipe, we'll look at how we can deploy a Silverlight application.

How to do it...

Deploying a Silverlight application is easy; the Silverlight code is compiled and wrapped into a *.xap file. Getting this file on the client side and running it from there is our only concern. The following steps are to be carried out for deploying a Silverlight application:

1. We'll use the `DeployHelloWorld` application to demonstrate deployment, which is available with the code downloads in the `Chapter01/DeployHelloWorld` folder. Build the application and notice that Visual Studio has created a *.xap file in the `ClientBin` directory. This file, which is nothing more than a *.zip file but with another extension, contains the assembly (one or more) to which our Silverlight application was compiled, optional resources, and the `AppManifest.xaml` file.
2. While looking at the files created by default by Visual Studio in the web project, a sample HTML (`DeployHelloWorldTestPage.html`) and ASPX (`DeployHelloWorldTestPage.aspx`) page are created for us as shown in the following screenshot:



3. Both pages have an OBJECT tag included. One of the parameters is named as Source and it has a reference to the *.xap file in the ClientBin as shown in the following code. If we want to deploy the *.xap file to another location, we need to update this reference. We'll use the default as of now.

```
<object data="data:application/x-silverlight-2,"
        type="application/x-silverlight-2"
        width="100%"
        height="100%">
  <param name="source" value="ClientBin/DeployHelloWorld.xap"/>
  <param name="onError" value="onSilverlightError" />
  <param name="background" value="white" />
  <param name="minRuntimeVersion" value="4.0.41108.0" />
  <param name="autoUpgrade" value="true" />
  <a href="http://go.microsoft.com/fwlink/?LinkID=149156&v=4.0.4
    1108.0"
    style="text-decoration:none">
    
  </a>
</object>
```

Note that the value of minRuntimeVersion may differ slightly because of different Silverlight version releases.

4. If using the HTML page, the following files need to be copied:
- DeployHelloWorldTestPage.html
 - Silverlight.js
 - ClientBin/DeployHelloWorld.xap
5. If using the ASPX page, we need to copy the following files:
- DeployHelloWorldTestPage.aspx
 - Silverlight.js
 - ClientBin/DeployHelloWorld.xap
 - bin directory, if using code-behind for the ASPX page
 - web.config
6. We'll need to test the page in a browser. If it fails to load, check the MIME types served by the web server software. There should be *.xap and *.xaml in there. (They are specified as the data type in the OBJECT tag.)

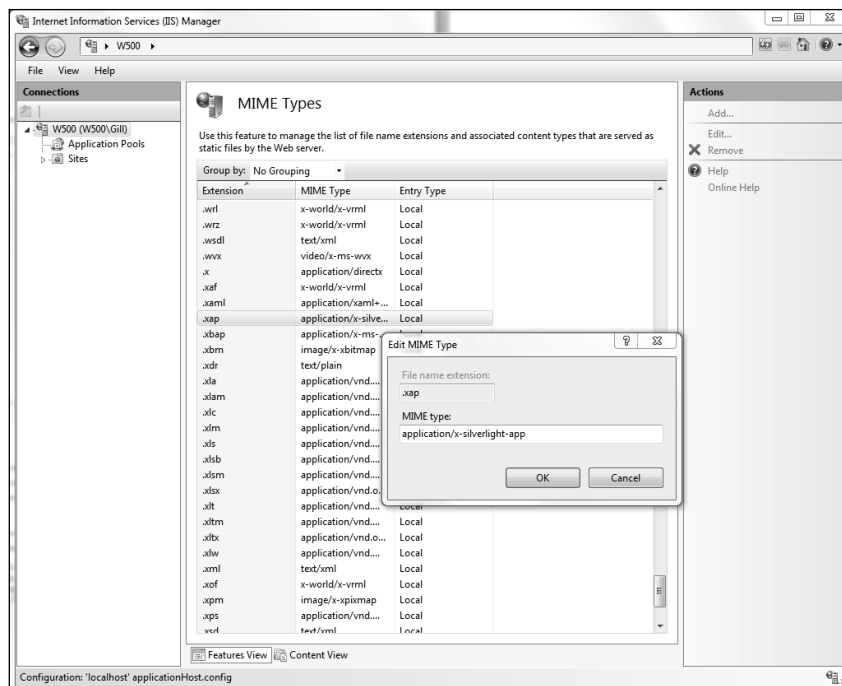
How it works...

One of the best things about Silverlight is that it can run from any type of server. If we're using ASP.NET, PHP, JSP, or plain-old HTML, Silverlight can still be embedded. Silverlight runs on the client side. The plugin has a CLR embedded so that it hosts our application. On the server side, the only thing we need to do is to serve the files (most importantly, the *.xap file) that will be downloaded to the client side when requested.

Configuration changes on the server

If the Silverlight application isn't being shown, it might be that the server software (IIS or Apache) is not configured to serve the file types used by Silverlight (*.xap and *.xaml). Windows Vista SP1 and Windows Server ship with **Internet Information Services (IIS 7)** while Windows 7 and Windows Server 2008 R2 include IIS 7.5. On these OS versions, both IIS 7 and IIS 7.5 are configured out of the box to serve *.xap and *.xaml files. On Windows Vista without SP1, we need to add these to the known MIME types. We can do this by opening **Internet Information Services (IIS) Manager** and selecting **MIME Types**. Then, we simply click on **Add** and add the following two items:

- ▶ .xap in the **File name extension:** field and application/x-silverlight-app in the **MIME type:** field
- ▶ .xaml in the **File name extension:** field and application/xaml+xml in the **MIME type:** field



What if the server doesn't allow using XAP?

If the server environment doesn't allow adding MIME types (a shared hosting plan), there's no reason to panic. As a *.xap file is nothing more than a *.zip file but with another extension, Silverlight supports the *.xap file being deployed as a *.zip file.

To get things working, start by renaming the *.xap file in the ClientBin to *.zip. Also, replace the reference to the *.xap file to the new name as shown in the following code:

```
<object data="data:application/x-silverlight-2,"
        type="application/x-silverlight-2"
        width="100%"
        height="100%">
  <param name="source" value="ClientBin/DeployHelloWorld.zip"/>
  ...
</object>
```