

8

Video Library: Silverlight 1.1 Case Example

For our Silverlight 1.1 example, we chose to port our Silverlight 1.0 example to 1.1. This provides a good feel for the differences between the two versions and for how to port applications from 1.0 to 1.1, and, in particular, it gives us an opportunity to see what is better about developing for 1.1. This chapter doesn't duplicate the in-depth explanation of Chapter 7; instead, it focuses on the main differences and changes between the 1.0 and 1.1 versions of the example application, Lumos. So it is recommended that you review Chapter 7 first to get familiarity with the solution.

You can view the Silverlight 1.1 version of the Lumos application online by visiting http://labs.infragistics.com/wrox/silverlight1_0/chapter8.

Also please note that the source code for the Lumos application is included along with the other code from the book and is available for download from www.wrox.com. However, to keep the file size of the download manageable, the code for the Lumos application available for download does not include the video clips from the full application. This means that the buttons will not work (due to missing videos), but it should be sufficient to get the idea across. You can download a couple of the videos separately, however, to see them in action.

Getting Started

To get started creating Lumos for Silverlight 1.1, you first need to assemble all of the development tools needed to build a Silverlight 1.1 application. The tools for building Silverlight 1.1 applications are a bit different from those needed for 1.0 development. The tools needed for building Lumos are:

- Microsoft Silverlight 1.1 Software Development Kit [Alpha Refresh or later]

Chapter 8: Video Library: Silverlight 1.1 Case Example

- ❑ Visual Studio 2008 [Beta 2 or later]
- ❑ Microsoft Silverlight Tools for Visual Studio [Alpha Refresh July 2007 or later]
- ❑ Expression Blend 2 [August 2007 Preview or later]
- ❑ Expression Encoder

Once you have all of the tools in place, you are ready to start building the application.

The screenshots in this chapter are taken from a virtual machine running Windows XP Professional SP2, so they differ from the rest of the book, which uses Vista. As a rule, we recommend using virtual machines for pre-release versions of Visual Studio in order to avoid any problems with it affecting your regular development (using earlier versions).

Key Changes between 1.0 and 1.1

Overall, porting to 1.1 was mostly the grunt work of converting the JavaScript to C#. Although you could, of course, use VB.NET, it was easier to go from JavaScript to C# due to their syntactic similarities. We made some refactoring to better encapsulate the controls, creating public properties, events, and public methods where we needed to provide control and notification to external instances, such as the page that uses the controls. The following sections discuss some of the key differences and changes we made in porting to 1.1.

Creating a 1.1 Silverlight Control Library

We created a `Wrox.Silverlight1_1.Controls` project. This is a Silverlight class library project (Figure 8-1), which is the “right” way to package up Silverlight controls for reuse because we want to be able to share these controls, in theory, across projects. We just put the `Button` and `VideoArea` controls in there because they could, theoretically, be generic.

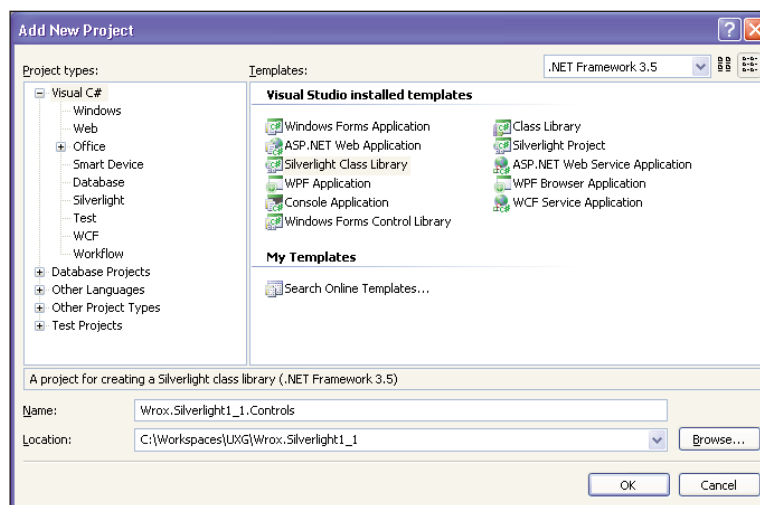


Figure 8-1

Chapter 8: Video Library: Silverlight 1.1 Case Example

Creating a 1.1 Silverlight Project

We created a `Wrox.Silverlight1_1.Lumos` project, as shown in [Figure 8-2](#). This is a regular Silverlight project (not a control library). The benefit here is the creation of “pages” for Silverlight. “Pages” (loose XAML files) are what you need to point the Silverlight control at. The Silverlight class library template bundles the XAML into the assembly, so it doesn’t work well for targeting. Here are a couple of other nice things about the Silverlight project template:

1. It gives you a nifty little test page.
2. It automatically creates class-level fields for all UI elements that have a name and hooks them up (in a partial class) so that you don’t manually have to do the whole `FindName` thing to get them yourself.

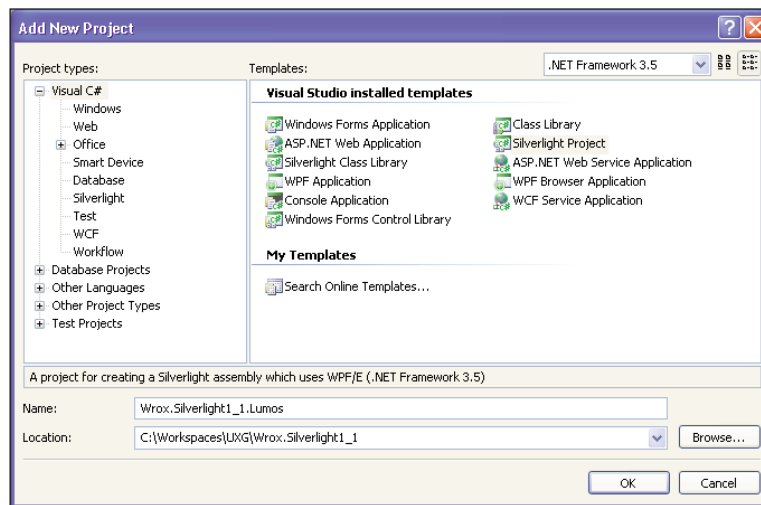


Figure 8-2

This project has both our main scene/page as well as the `ComboBox` control. You’ll notice that there is no longer any use of `createFromXaml` as we had in JavaScript. Where the 1.0 app was using that, we either moved the XAML into an XAML file or refactored it into a control. Another key change is that the controls don’t have/require knowledge outside of themselves. The way they were done in the 1.0 app, you had the video area knowing about the buttons and the buttons knowing about the video area. It’s best that controls don’t know about each other as a rule — the code that glues them together goes in the page or in a higher-level composite control. In our case, we refactored the code such that the glue code lives in the main scene/page. This meant creating a couple of public events in the controls to let the page know when stuff happened.

Creating the New Lumos Web Site

We created the new Lumos web site ([Figure 8-3](#)). It’s just a regular ASP.NET 3.5 web site. Everything happens in managed, client-side code, either in the controls or in the main page/scene. The only thing the web site really does is host the web service, which we renamed and moved to follow best practices.

Chapter 8: Video Library: Silverlight 1.1 Case Example

The reason this is a good practice is that it makes a good namespace to put all of your services in going forward (not just the one you have now), and “Services” is a good name for the space given that it contains application services.

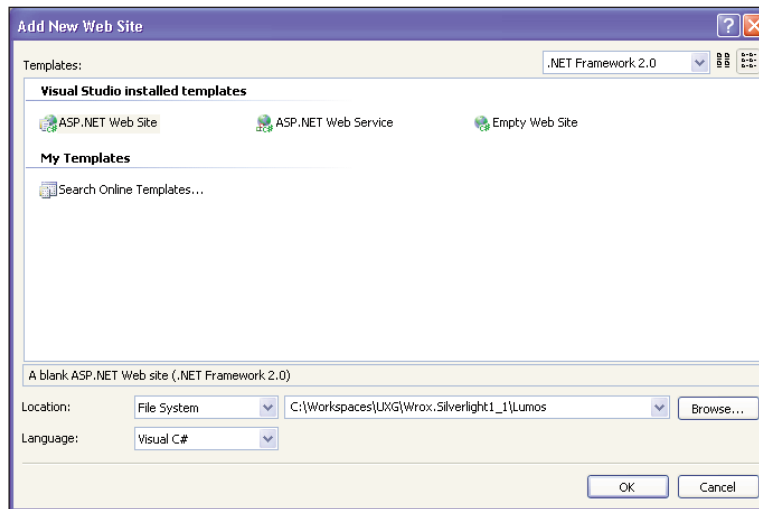


Figure 8-3

The one thing to note is that when you have an ASP.NET web site that needs to use a Silverlight project (control library or regular), you use the Add Silverlight Link context menu item on the web site (Figure 8-4) and choose the Silverlight project(s) as in Figure 8-5; we did that for both Silverlight projects. What this does is automatically copy over the project output assemblies into a ClientBin folder as well as copy over any “pages” so that you can reference them using `createSilverlightEx` on the Silverlight control.

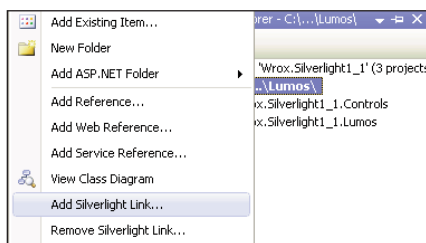


Figure 8-4

Chapter 8: Video Library: Silverlight 1.1 Case Example

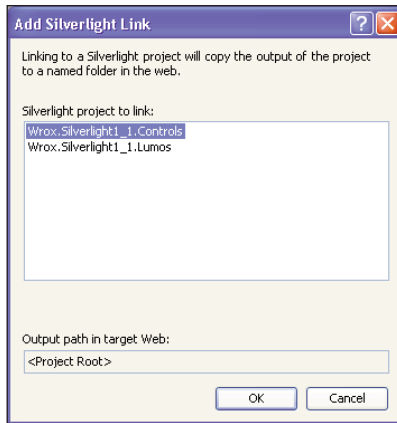


Figure 8-5

You can see the results of this in Figure 8-6. Note the `ClientBin` folder. When you compile the Silverlight projects, the output (DLL and PDB, if applicable) will be copied over to the `ClientBin` of the web site. Also, note that `Page.xaml` (which we cover later) is the only “page” in our Silverlight project, so it is copied over to the web site as well.

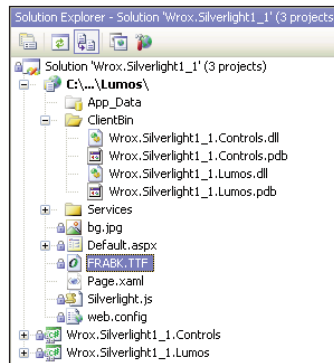


Figure 8-6

Converting JavaScript to C#

The most labor-intensive part of porting was, as you might expect, converting the JavaScript to C#. Thankfully, much of the syntax is similar, but that helped only so much because C# is strongly typed and JavaScript is not. Also, even though the Silverlight control is forgiving about case, C# is not, and much of the 1.0 code was inconsistent in terms of casing. We definitely recommend trying to follow the casing of the actual control properties in your 1.0 code if you think you might ever port it.

Chapter 8: Video Library: Silverlight 1.1 Case Example

Using `FindName` Less

One of the key differences between coding in 1.1 controls and pages versus the 1.0 implementation is that you use `FindName` far less — you typically (with Silverlight controls) use it up front to find the relevant XAML elements and store the reference at the control class level. With pages, VS does this for you; so you can just use the `x:Name` that you give the element in the XAML to reference it in the code behind.

So, for example, the ported code for the `VideoArea` control starts like this:

```
public class VideoArea : XamlControl
{
    #region Fields & Properties
    #region Controls
    Canvas _Play;
    Canvas _Stop;
    Canvas _Pause;
    Canvas _Previous;
    Canvas _Next;
    Canvas _PositionHandle;

    MediaElement _Player;
    /// <summary>
    /// Gets the media player for this instance.
    /// </summary>
    public MediaElement Player
    {
        get
        {
            return _Player;
        }
    }

    Canvas _Track;
    Storyboard _PlayTimer; // emulated timer

    TextBlock _Text;
    TextBlock _Description;
}
```

We set up class-level fields for the various UI elements that we want to reference in code, and then we find them (only once) using `FindName` in the constructor after initializing the XAML. One of the oddities in using `FindName` is that we have to keep a reference to the parsed XAML for the control and use `FindName` on it rather than, say, using `this.FindName` directly (which will never find anything).

You'll notice when you create a new Silverlight control that it uses `Assembly.GetManifestResourceStream` to get the embedded XAML and uses the `InitializeFromXaml` method (inherited from its base, `System.Windows.Controls.Control`). The thing is that you need to keep a reference to the returned `FrameworkElement` and use that instance's `FindName` in order to successfully find elements from your XAML. So what we did in order to minimize code duplication (and because it is generally good to inherit from your own base class to give yourself a point of shared inheritance in your own library) is create a base `XamlControl` class.

Listing 8-1 shows the complete `XamlControl` class. We're able to centralize the initialization by using the `type.FullName` member of the instance's type and appending `".xaml"` to it. Because

Chapter 8: Video Library: Silverlight 1.1 Case Example

GetManifestResourceStream expects the full namespace plus filename of where the file is located in the project, this should always work. We then store the parsed XAML FrameworkElement in our `_Root` field and expose a read-only property, `RootElement`, so that inheritors can use that to find UI elements in the XAML.

Listing 8-1: The Base XamlControl Class

```
namespace Wrox.Silverlight1_1.Controls
{
    /// <summary>
    /// Base class for our controls.
    /// </summary>
    public class XamlControl : Control
    {
        #region Fields & Properties
        FrameworkElement _Root;
        /// <summary>
        /// Gets the root element from the Xaml.
        /// </summary>
        public FrameworkElement RootElement
        {
            get
            {
                return _Root;
            }
        }
        #endregion

        #region Constructors
        /// <summary>
        /// Reads the Xaml for the control into
        /// <see cref="RootElement"/>.
        /// </summary>
        public XamlControl()
        {
            this.Init();
        }
        #endregion

        #region Methods
        void Init()
        {
            Type type = this.GetType();
            System.IO.Stream s =
                type.Assembly.GetManifestResourceStream(
                    type.FullName + ".xaml");
            _Root = this.InitializeFromXaml(
                new System.IO.StreamReader(s).ReadToEnd());
        }
        #endregion
    }
}
```

Chapter 8: Video Library: Silverlight 1.1 Case Example

As mentioned, taking this approach enables our deriving classes to, first of all, not have to have the initialization code repeated, and also provides the `RootElement` property to use in finding elements, as shown next from the `VideoArea` control:

```
_PositionHandle =
    (Canvas)this.RootElement.FindName("positionHandle");
_PositionHandle.ReleaseMouseCapture();
_PositionHandle.MouseMove +=
    new MouseEventHandler(this.PositionHandleMouseMove);
```

This code illustrates how you can then use `FindName` (like you do in 1.0) to find your XAML element. The difference is that you need to use the `RootElement` and that, because you're now in a strongly typed environment, you have to cast the result. Once you find the element, though, you're free to use the reference anywhere in the class and don't have to keep using `FindName` to get to it.

Thankfully, as we mentioned, Visual Studio takes care of the whole initialization and `FindName` stuff in regular Silverlight projects using the Silverlight Page item template. So you can just start using your elements right away in your code.

Missing Timer

Another fun thing was getting around the missing timer in 1.1. The 1.0 code was using JavaScript's `setInterval` to update the play position on the track. Silverlight 1.1 doesn't have an equivalent, so the current workaround is to use a `Storyboard` with a double animation and just restart it in its completed handler. In the 1.1 Alpha Refresh, double animations require setting the target object and target property; so we also had to add a hidden rectangle for it to target, even though it doesn't really do anything to it or show at all.

Here's the XAML for the `Storyboard`:

```
<Canvas.Resources>
  <Storyboard x:Name="timer">
    <DoubleAnimation
      x:Name="animation"
      Duration="00:00:0.5"
      Storyboard.TargetName="InvisibleRect"
      Storyboard.TargetProperty="Width" />
  </Storyboard>
</Canvas.Resources>
<Rectangle Visibility="Collapsed" x:Name="InvisibleRect" />
```

In the code, we get the reference like so:

```
_PlayTimer =
    (Storyboard)this.RootElement.FindName("timer");
this._PlayTimer.Completed +=
    new EventHandler(_PlayTimer_Completed);
```

And in the `_PlayTimer_Completed` handler, we just update the trackbar position and then restart the `Storyboard`. Note that we don't set the `Storyboard` to repeat forever. If you do this, you don't get the completed event raised; so you have to let it end, do your stuff, and restart it.

Chapter 8: Video Library: Silverlight 1.1 Case Example

We should get a timer before 1.1 RTM, so this is only a temporary workaround until we get that.

```
void _PlayTimer_Completed(object sender, EventArgs e)
{
    this.UpdateTrackBarPosition();
    this._PlayTimer.Begin();
}
```

Strong Typing and Media Positioning

The “position.seconds” stuff from the 1.0 version had to be changed because Position is a TimeSpan in 1.1, and you can't just use Position.Seconds — you have to use Position.TotalSeconds; otherwise, you get some goofy behavior on the trackbar because the Seconds property is just the fractional seconds. Similarly, when setting it, you have to use TimeSpan.FromSeconds. To give you a feel for this, consider the ported code to update the trackbar location:

```
void UpdateTrackBarPosition()
{
    //don't update the handle if it's captured
    if (_PositionHandle.CaptureMouse())
    {
        return;
    }

    //get the track width and media duration
    double trackwidth = _Track.Width - 14;
    double duration =
        _Player.NaturalDuration.TimeSpan.TotalSeconds;

    if (duration <= 0)
    {
        return;
    }

    //determine the current percentage that has been played
    double playpercent =
        _Player.Position.TotalSeconds / duration;

    //translate the percentage played into a position on the track
    double handleposition = trackwidth * playpercent;

    //set the handle to the new position
    _PositionHandle.SetValue(Canvas.LeftProperty,
        DEFAULT_HANDLE_LEFT + handleposition);
}
```

Another interesting thing to note is that when using attached properties, you actually pass in a reference to the attached property rather than using the string name for it (as seen in the last line in the preceding code block).

Chapter 8: Video Library: Silverlight 1.1 Case Example

Specifying Colors in Code

As far as we can tell, there's no `Color.FromWeb` where you can pass the hex-based string that is common on the web (and you can use it in XAML). Maybe there is one and we just missed it, but we ended up using Blend to get the RGB values for the coded colors and using `Color.FromRgb` to get them. It's annoying, and we hope that'll be fixed by RTM. Here's an example from the `MouseLeftButtonUp` handler in the `Button` control:

```
this._Background.Fill =
    GetGradient(Color.FromRgb(227, 241, 255),
        Color.FromRgb(93, 145, 185));
```

Again, we just use the `Color.FromRgb` method and pass in the numeric values for the RGB. It seems like there must be a way to do this using the hex value because you can specify hex in XAML. (You can use the .NET hex notation as well, but it is still not very familiar to most devs.)

Better Encapsulation and Reuse

We encapsulated control members and provided control to consumers (as you should with controls — not requiring or allowing consumers to know or use internal members). For instance, we created the public `MediaPlayer` property on the `VideoArea` control to allow consumers to interact directly with the `MediaElement` control that we use to play the videos. We created the `MediaOpened` event that is raised when the media is changed to enable the page to react as needed and coordinate changes to other controls, and we added the `PlayMedia` method to allow consumers to directly play media in the video area control.

Along with this, thanks to the better control model in 1.1, we could ditch all usage of string replacement to customize an instance of the control. Whereas in 1.0 we had to reuse just the XAML and replace IDs and such as needed, in 1.1 we can just create instances of our controls using the control constructor, or we can declare them in XAML. This is a much more familiar way of using controls than the kind of hacky approach we have in 1.0.

Here's an example of using the `VideoArea` control in our `Page.xaml`. First of all, you have to declare an XML namespace (similar to registering a tag prefix in ASP.NET):

```
xmlns:wrox="clr-namespace:Wrox.Silverlight1_1.Controls;assembly=ClientBin/
Wrox.Silverlight1_1.Controls.dll"
```

You can create any XML namespace prefix (not already used) that you like for your controls. We picked "wrox." Then you just use the `clr-namespace:<namespace>;assembly=<relative assembly path>` format to tell Silverlight where to find the controls in that CLR/.NET namespace. This should be very familiar to those using WPF. Once you have that, you use the control just like you would any other — you just prefix the control name with your namespace prefix:

```
<wrox:VideoArea x:Name="VideoArea" />
```

Chapter 8: Video Library: Silverlight 1.1 Case Example

You can, of course, set any of the public properties in the declaration just as you normally would. In the page's code behind, you can then reference it using the name you gave it, for example, `this.VideoArea`. To use a new control in code, as mentioned, you just use the control constructor like so:

```
Wrox.Silverlight1_1.Controls.Button button =
    new Wrox.Silverlight1_1.Controls.Button();
```

Then you can access whatever public properties it has, as in the following excerpt from our page's `OnVideosDownload` event handler:

```
button.SetValue<double>(Canvas.TopProperty, top);
button.Text = vid.Title;
button.Value = (i+1).ToString();
button.Tag = vid;
button.Clicked += new MouseEventHandler(button_Clicked);
```

One of the benefits of our approach here is that we created a new `Tag` property on our button that is of type `Object`. This enables us to store a direct reference to the `Video` instance related to that particular button — when the button is clicked, we just access that property to get the selected video:

```
void button_Clicked(object sender, MouseEventArgs e)
{
    Wrox.Silverlight1_1.Controls.Button button =
        (Wrox.Silverlight1_1.Controls.Button)sender;
    Services.Video vid = (Services.Video)button.Tag;
    this.CurrentVideoTitle.Text = vid.Title;
    this.Combo.SetValue<double>(Canvas.LeftProperty,
        (this.CurrentVideoTitle.ActualWidth +
        (double)this.CurrentVideoTitle
            .GetValue(Canvas.LeftProperty) + 10));
    this.Combo.Visibility = Visibility.Visible;
    this.ComboBody.Children.Clear();
    this.VideoArea.PlayMedia(
        new Uri(vid.Uri, UriKind.Relative),
        vid.Description);
}
```

As you can see, we just cast the sender as a `Button` and retrieve the `Video` instance attached to it using the `Tag` property. Once we have that, we just set up the page to play the video using the information stored on the `Video` instance (that was retrieved from the web service).

Also, because of the better encapsulation, we're able to cut out using the Silverlight downloader to retrieve the XAML templates and all of the code that was used to customize them. This left us with only needing to call our web service.

Using Web Services

On the ASP.NET side, we didn't change much. We just moved the service into a `Services` directory and gave it a friendly name. Then all we had to do was add a web reference in the Silverlight project. Right-click the Silverlight project and choose `Add Web Reference`. Then choose `Web Services` in this Solution on the dialog that pops up. You should then see the dialog shown in [Figure 8-7](#).

Chapter 8: Video Library: Silverlight 1.1 Case Example

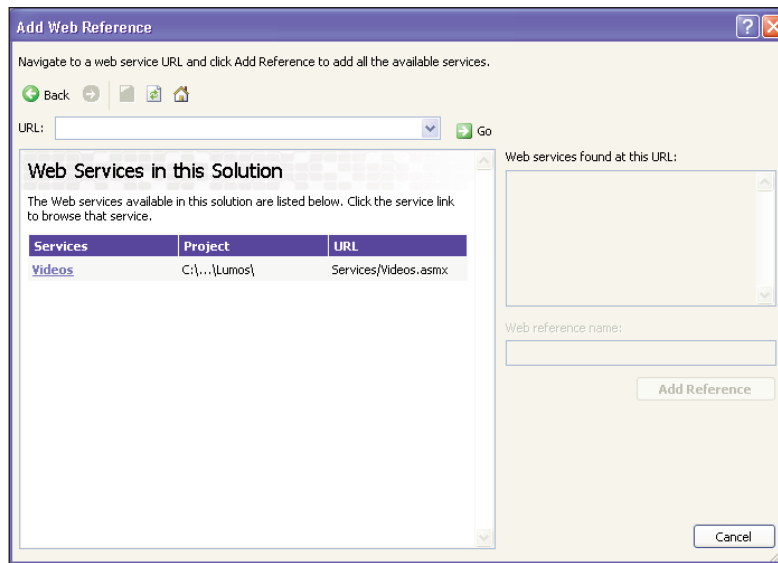


Figure 8-7

You select the service and give it a name of `Services`. Doing this gives it the full namespace of `Wrox.Silverlight1_1.Lumos.Services` because of the way the Add Web Reference tool works (it prepends your root namespace). This makes it nice to use in code as we do in our `Page` class.

```

Services.LumosVideos _VideoService;
/// <summary>
/// Gets the Lumos videos service.
/// </summary>
public Services.LumosVideos VideoService
{
    get
    {
        if (_VideoService == null)
        {
            _VideoService =
                new Services.LumosVideos();
        }
        return _VideoService;
    }
}

```

By doing this, we can simply use `this.VideoService` in our code to call methods on the web service, as we do in our `Page_Loaded` event handler. We get a nifty `Page_Loaded` event handler that gets hooked up via its being declared in the XAML, so we can do our page initialization code in it. Note the call to `InitializeComponent` — this is where the VS-generated code does the grunt work of finding our UI elements and hooking them up to class (page) level fields. Then we can just use them as we do here:

```

public void Page_Loaded(object o, EventArgs e)
{

```

Chapter 8: Video Library: Silverlight 1.1 Case Example

```

// Required to initialize variables
InitializeComponent();

// start getting videos
this.VideoService.BeginGetVideos(
    new AsyncCallback(this.OnVideosDownload), null);

this.Combo.MouseLeftButtonDown +=
    new MouseEventHandler(Combo_MouseLeftButtonDown);
this.Combo.MouseLeftButtonUp +=
    new MouseEventHandler(Combo_MouseLeftButtonUp);
this.ComboBody.MouseLeftButtonUp +=
    new MouseEventHandler(ComboBody_MouseLeftButtonUp);
this.VideoArea.MediaOpened +=
    new EventHandler(VideoArea_MediaOpened);
}

```

Note that `this.Combo`, `this.ComboBody`, and `this.VideoArea` are all automatically set up for us as references to the corresponding UI elements in the XAML. Here we're just hooking up our event handlers. The only other thing we do in the page loading is to start an asynchronous call to our video service's `GetVideos` method (using `BeginGetVideos`). The handler for that follows:

```

void OnVideosDownload(IAsyncResult result)
{
    try
    {
        double top = DEFAULT_TOP;
        Services.Video[] videos =
            this.VideoService.EndGetVideos(result);
        for (int i = 0; i < videos.Length; i++)
        {
            Services.Video vid = videos[i];
            Wrox.Silverlight1_1.Controls.Button button =
                new Wrox.Silverlight1_1.Controls.Button();
            button.SetValue<double>(Canvas.TopProperty, top);
            button.Text = vid.Title;
            button.Value = (i+1).ToString();
            button.Tag = vid;
            button.Clicked += new MouseEventHandler(button_Clicked);
            this.Menu.Children.Add(button);
            top += 60;
        }
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine(ex.ToString());
        throw;
    }
}

```

In the current release, we're still limited to using the `Canvas` element as our only layout, so we're still stuck with absolute positioning just like in 1.0. This code is more or less equivalent to the 1.0 version — for each video returned from our video service, we create a button to represent it in our left menu. As you can see, using web services in Silverlight 1.1 is a breeze.

Chapter 8: Video Library: Silverlight 1.1 Case Example

The only other significant code in the page has to do with setting up the markers for the combo box at the top. That code is basically the same as in 1.0 (just converted).

Summary

Porting an application from Silverlight 1.0 to Silverlight 1.1 is a labor-intensive process, but you can make it easier by doing your best to mimic 1.1 development in 1.0. You can't totally get away from the lack of actual controls in 1.0, but if you follow the approaches outlined in Chapter 5, you'll see ways that you can emulate having controls in JavaScript. If you follow that approach and adhere to the actual casing of the Silverlight control's methods and properties, you'll save yourself pain in porting.

This chapter covered the key differences between developing the Lumos application in 1.0 and developing it in 1.1. It also suggested the best way to create Silverlight applications by using control libraries, Silverlight projects, and integrating into an ASP.NET application with web services. By exploring this chapter and the accompanying code, you should get a very solid idea of how to go about creating applications for Silverlight 1.1.

Naturally, as Silverlight 1.1 develops, a lot of this will change, and as features are added, there will be much more to say about it. We'll do our best to keep our sample up to date or, if it seems good, we may create a new 1.1 application that better takes advantage of the features that end up being added to the 1.1 framework. As Chapter 6 described, there are plenty of areas where 1.1 can improve, and so far, Microsoft has indicated its intent to add features in most, if not all, of those areas in order to make Silverlight 1.1 a truly great platform for developing rich, cross-platform Internet applications.