Silverlight and Rich Internet Applications

29

Had I the heavens' embroidered cloths, Enwrought with gold and silver light.
—William Butler Yeats

This world is but a canvas to our imaginations.
—Henry David Thoreau

Something deeply hidden had to be behind things.
—Albert Einstein

Individuality of expression is the beginning and end of all art.
—Johann Wolfgang von Goethe

Objectives

In this chapter you'll learn:

- How Silverlight relates to WPF.
- To use Silverlight controls to create Rich Internet Applications.
- To create custom Silverlight controls.
- To use animation for enhanced GUIs.
- To display and manipulate images.
- To use Silverlight with Flickr's web services to build an online photo-searching application.
- To create Silverlight deep zoom applications.
- To include audio and video in Silverlight applications.

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

Outline

29.1 Introduction
29.2 Platform Overview
29.3 Silverlight Runtime and Tools Installation
29.4 Building a Silverlight **WeatherViewer** Application
29.4.1 GUI Layout
29.4.2 Obtaining and Displaying Weather Forecast Data
29.4.3 Custom Controls
29.5 Animations and the **FlickrViewer**
29.6 Images and Deep Zoom
29.6.1 Getting Started With Deep Zoom Composer
29.6.2 Creating a Silverlight Deep Zoom Application
29.7 Audio and Video
29.8 Wrap-Up

Summary | *Terminology* | *Self-Review Exercises* | *Answers to Self-Review Exercises* | *Exercises*

29.1 Introduction

Silverlight™ is Microsoft's platform for building **Rich Internet Applications** (**RIAs**)—web applications comparable in responsiveness and rich user interactivity to desktop applications. Silverlight is a robust, cross-platform, cross-browser implementation of the .NET platform that competes with RIA technologies such as Adobe Flash and Flex and Sun's JavaFX, and complements Microsoft's ASP.NET and ASP.NET AJAX (which we discussed in Chapter 27). Developers familiar with programming WPF applications are able to adapt quickly to creating Silverlight applications.

The "sizzle" of Silverlight is **multimedia**—the use of images, graphics, animation, sound and video to make applications "come alive." Silverlight includes strong multimedia support, including state-of-the-art high-definition video streaming. WPF and Silverlight, through the .NET class libraries, provide extensive multimedia facilities that enable you to start developing powerful multimedia applications immediately. Among these facilities is **deep zoom**, which allows the user to view high-resolution images over the web as if the images were stored on the local computer. Users can interactively "explore" a high-resolution image by zooming in and out and panning—while maintaining the original image's quality. Silverlight supports deep zoom images up to one billion by one billion pixels in size!

[*Note:* The **WeatherViewer** and **FlickrViewer** examples require web service API keys from WeatherBug and Flickr, respectively, before you can execute them. See Sections 29.4–29.5 for details.]

29.2 Platform Overview

Silverlight runs as a browser plug-in for Internet Explorer, Firefox and Safari on recent versions of Microsoft Windows and Mac OS X. The system requirements for the runtime can be found at `www.microsoft.com/silverlight/faq/#sys-req`. Silverlight is also available on Linux systems via the Mono Project's Moonlight (`mono-project.com/Moonlight`).

Like WPF applications, Silverlight applications consist of user interfaces described in XAML and code-behind files containing application logic. The XAML used in Silverlight is a subset of that used in WPF.

The subset of the .NET Framework available in Silverlight includes APIs for collections, input/output, generics, multithreading, globalization, XML, LINQ and more. It

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

also includes APIs for interacting with JavaScript and the elements in a web page, and APIs for local storage data to help you create more robust web-based applications.

Silverlight is an implementation of the .NET Platform, so you can create Silverlight applications in .NET languages such as Visual C#, Visual Basic, IronRuby and IronPython. This makes it easy for .NET programmers to create applications that run in a web browser.

Silverlight's graphics and GUI capabilities are a subset of the Windows Presentation Foundation (WPF) framework. Some capabilities supported in Silverlight include GUI controls, layout management, graphics, animation and multimedia. There are also styles and template-based "skinning" capabilities to manage the look-and-feel of a Silverlight user interface. Like WPF, Silverlight provides a powerful data-binding model that makes it easy to display data from objects, collections, databases, XML and even other GUI controls. Silverlight also provides rich networking support, enabling you to write browser-based applications that invoke web services and use other networking technologies.

29.3 Silverlight Runtime and Tools Installation

Silverlight runs in web browsers as a plug-in. To view websites programmed in Silverlight, you need the **Silverlight Runtime** plug-in from `www.silverlight.net/getstarted/`. After installing the plug-in, go to Microsoft's Silverlight Showcase (`www.silverlight.net/showcase/`) to try some sample applications.

The examples in this chapter were built using the Silverlight 4 SDK, which is available from

```
bit.ly/SilverlightDownload
```

and Visual Web Developer 2010 Express, which is available from:

```
www.microsoft.com/express/web/
```

Additional information about Silverlight is available at:

```
www.silverlight.net
```

29.4 Building a Silverlight WeatherViewer Application

Silverlight is a subset of WPF, so the two share many capabilities. Since Silverlight produces Internet applications instead of desktop applications, the setup of a Silverlight project is different from that of WPF.

A Silverlight application created with the **Silverlight Application** project template has two XAML files—`MainPage.xaml` and `App.xaml`. `MainPage.xaml` defines the application's GUI, and its code-behind file `MainPage.xaml.cs` declares the GUI event handlers and other methods required by the application. `App.xaml` declares your application's shared resources that can be applied to various GUI elements. The code-behind file `App.xaml.cs` defines application-level event handlers, such as an event handler for unhandled exceptions. Content in the `App.xaml` and `App.xaml.cs` files can be used by all the application's pages. We do not use `App.xaml` and `App.xaml` in this chapter. In Visual Web Developer 2010 and Visual Studio 2010 there is also a **Silverlight Navigation Application** project template for creating multipage Silverlight applications. We do not cover this template.

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

Differences Between WPF and Silverlight

To create a new Silverlight project in Visual Web Developer Express, select **File > New Project...**. In the **Installed Templates** pane under **Visual C#**, select the **Silverlight** option. Then in the **Templates** window, select **Silverlight Application**. After entering your project's name (`WeatherViewer`) and selecting its location, click **OK**. A **New Silverlight Application** dialog appears, asking how you would like to host your application. Ensure that the **Host the Silverlight application in a new Web site** option is selected. In the **New Web project type** drop-down menu, select **ASP.NET Web Application Project**. Keep the default project name and click **OK**.

`MainPage.xaml`

The `MainPage.xaml` file displayed in the XAML tab of Visual Studio (Fig. 29.1) is similar to the default XAML file for a WPF application. In a WPF application, the root XAML element is a `Window`. In Silverlight, the root element is a **`UserControl`**. The default `UserControl` has a class name specified with the **`x:Class`** attribute (line 1), specifies the namespaces (lines 2–5) to provide access to the Silverlight controls throughout the XAML, and has a width and height of 400 and 300 pixels, respectively. These numbers are system-independent pixel measurements, where each pixel represents 1/96th of an inch. Lines 9–11 are the default `Grid` layout container. Unlike a WPF application, the default `Grid`'s **`x:Name`** (the name used in code that manipulates the control) and `Background` attributes are set by default in a Silverlight application.

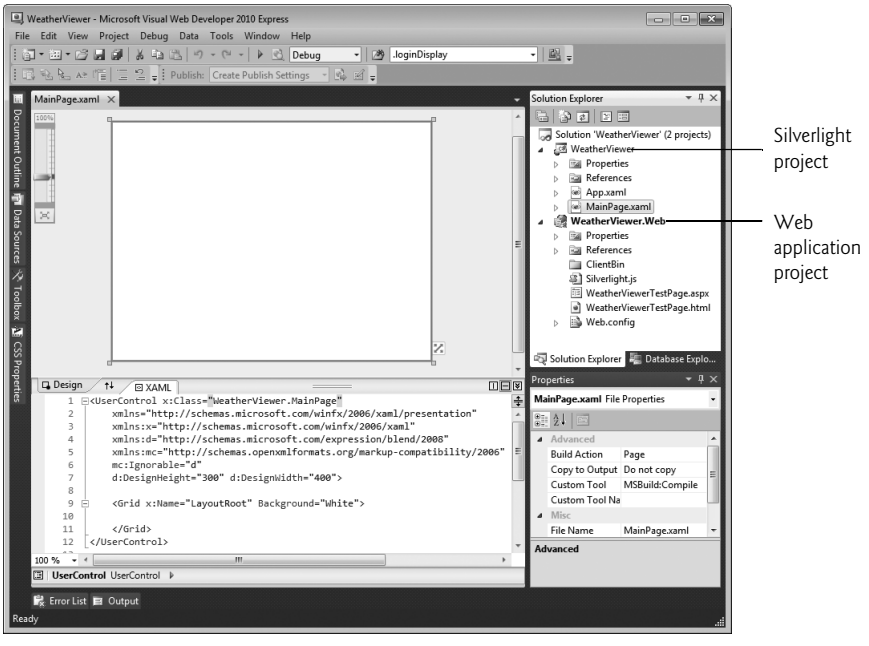


Fig. 29.1 | New Silverlight application in Visual Studio.

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

`.xap` File

A compiled Silverlight application is packaged by the IDE as a `.xap` file containing the application and its supporting resources (such as images or other files used by the application). The web page that hosts the Silverlight application references the Silverlight plug-in and the application's `.xap` file. The Silverlight plug-in then executes the application. The test web application that was created for you contains the file `WeatherViewerTestPage.aspx`, which loads and executes the Silverlight application.

A Silverlight application must be hosted in a web page. The **Web Application Project** is used to test the Silverlight application in a web browser. Building the solution automatically copies the compiled application into the **Web Application Project**. You can then test it using the built-in web server in Visual Studio. After the application is built in the IDE, this part of the application contains the `.xap` file that was described in the preceding paragraph.

Desiging Silverlight User Interfaces

As in WPF, you can use the **Design** view and the **Properties** window to design your user Silverlight interfaces, but the Visual Studio designer for WPF and Silverlight is not as robust as that provided by Microsoft Expression Blend. Expression Blend is beyond the scope of this chapter. Trial versions are available from `www.microsoft.com/expression/`.

Introduction to the WeatherViewer Application

Our **WeatherViewer** application (Fig. 29.2) allows the user to input a zip code and invokes a web service to get weather information for that location. The application receives weather data from `www.weatherbug.com`—a website that offers a number of weather-related web services, including some that return XML data. To run this example on your computer, you need to register for your own WeatherBug API key at `weather.weatherbug.com/desktop-weather/api.html`. This application uses LINQ to XML to process the weather data that is returned by the web service. The application also includes a custom control that we designed to display more detailed weather information for a day of the week selected by the user. Figure 29.2 shows the application after the user enters a zip code (Fig. 29.2(a)) then clicks Monday to see its weather details (Fig. 29.2(b)).

a)

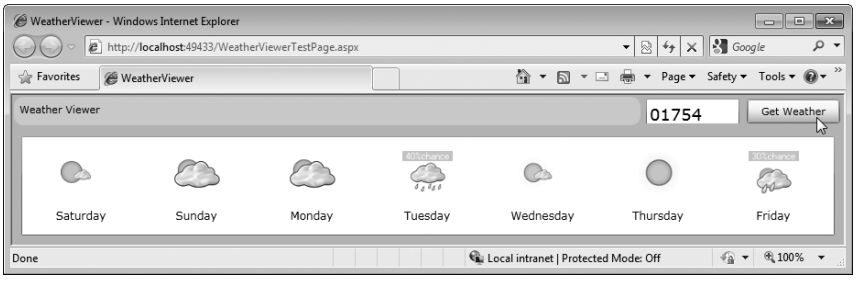


Fig. 29.2 | **WeatherViewer** application displays a seven-day weather forecast. The program can also display detailed information for a selected day. (Part 1 of 2.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

b)

Fig. 29.2 | **WeatherViewer** application displays a seven-day weather forecast. The program can also display detailed information for a selected day. (Part 2 of 2.)

29.4.1 GUI Layout

The layout controls of WPF described in Chapter 24—`Grid`, `StackPanel` and `Canvas`—are also available in Silverlight. The XAML for the layout of the **WeatherViewer** application is shown in Fig. 29.3. This application uses nested `Grid` controls to lay out its elements.

```
1  <!-- Fig. 29.3: MainPage.xaml -->
2  <!-- WeatherViewer displays day-by-day weather data (XAML). -->
3  <UserControl xmlns:Weather="clr-namespace:WeatherViewer"
4     x:Class="WeatherViewer.MainPage"
5     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
6     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
7     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
8     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
9     mc:Ignorable="d">
10
11    <Grid x:Name="LayoutRoot" Background="LightSkyBlue">
12       <Grid.RowDefinitions>
13          <RowDefinition Height="35" />
14          <RowDefinition x:Name="messageRow" Height="0" />
15          <RowDefinition />
16       </Grid.RowDefinitions>
17
18       <!-- Grid contains border, textbox and search button -->
19       <Grid>
20          <Grid.ColumnDefinitions>
21             <ColumnDefinition />
22             <ColumnDefinition Width="110" />
23             <ColumnDefinition Width="110" />
24          </Grid.ColumnDefinitions>
25
```

Fig. 29.3 | **WeatherViewer** displays day-by-day weather data (XAML). (Part 1 of 3.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
26          <!-- Border containing the title "Weather Viewer" -->
27          <Border Grid.Column="0" CornerRadius="10"
28             Background="LightGray" Margin="2">
29             <TextBlock Text="Weather Viewer" Padding="6" />
30          </Border>
31
32          <!-- zip code goes into this text box -->
33          <TextBox x:Name="inputTextBox" Grid.Column="1" FontSize="18"
34             Margin="4" Height="40"
35             TextChanged="inputTextBox_TextChanged" />
36
37          <!-- Click to invoke web service -->
38          <Button x:Name="getWeatherButton" Content="Get Weather"
39             Grid.Column="2" Margin="4" Click="getWeatherButton_Click" />
40       </Grid>
41
42       <!-- Border to contain text block which shows error messages -->
43       <Border x:Name="messageBorder" Background="White"
44          Grid.Row="1" Padding="8">
45          <TextBlock x:Name="messageBlock" FontSize="14"
46             HorizontalAlignment="Left" Foreground="Red"/>
47       </Border>
48
49       <!-- Contains weather images for several upcoming days -->
50       <ListBox x:Name="forecastList" Grid.Row="2" Margin="10"
51          SelectionChanged="forecastList_SelectionChanged">
52          <ListBox.ItemsPanel>
53             <ItemsPanelTemplate>
54                <!-- Arrange items horizontally -->
55                <StackPanel Orientation="Horizontal" />
56             </ItemsPanelTemplate>
57          </ListBox.ItemsPanel>
58
59          <ListBox.ItemTemplate>
60             <DataTemplate>
61                <!-- Represents item for a single day -->
62                <StackPanel Width="120" Orientation="Vertical"
63                   HorizontalAlignment="Center">
64                   <!-- Displays image for a single day -->
65                   <Image Source="{Binding WeatherImage}"
66                      Margin="5" Width="55" Height="58" />
67
68                   <!-- Displays the day of the week -->
69                   <TextBlock Text="{Binding DayOfWeek}"
70                      TextAlignment="Center" FontSize="12"
71                      Margin="5" TextWrapping="Wrap" />
72                </StackPanel>
73             </DataTemplate>
74          </ListBox.ItemTemplate>
75       </ListBox>
76
```

Fig. 29.3 | **WeatherViewer** displays day-by-day weather data (XAML). (Part 2 of 3.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
77          <!-- Custom control for displaying detailed information -->
78          <Weather:WeatherDetailsView x:Name="completeDetails"
79             Visibility="Collapsed" Grid.RowSpan="3" />
80       </Grid>
81    </UserControl>
```

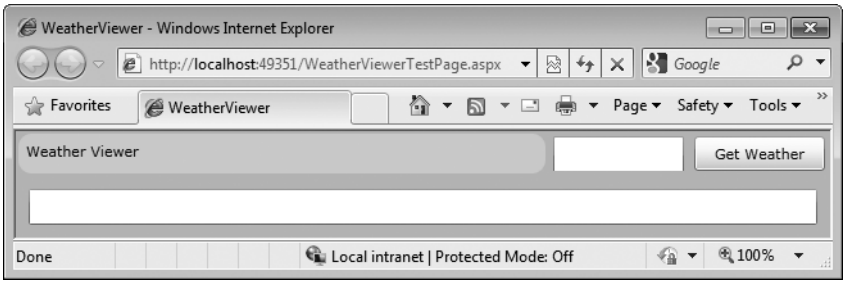


Fig. 29.3 | **WeatherViewer** displays day-by-day weather data (XAML). (Part 3 of 3.)

Lines 12–16 contain the `RowDefinitions` of the main `Grid`. Lines 20–24 contain the `ColumnDefinitions` of a nested `Grid` which displays the top row of the page containing the light gray title `Border`, the search `TextBox` and the search `Button`, as shown in Fig. 29.3.

Line 28 defines the `Border`'s **`Margin`** property, which specifies the amount of space between the `Border` and any adjacent elements. Lines 43–47 define a `Border` that contains a `TextBlock` in which we display an error message if the user enters an invalid zip code. Lines 49–75 define the `ListBox` used on the main page to display each day's weather image. Line 55 defines the `StackPanel` that is used as a template by the `ListBox`'s `ItemsPanel`, allowing the `ListBox`'s items to display horizontally. Lines 62–72 define a `StackPanel` for each individual item, displaying the weather `Image` and the `TextBlock` containing the day of the week in a vertical orientation. Lines 65 and 69 bind data from the web service's XML response to the two elements that display weather information.

Lines 78–79 create a `WeatherDetailsView` custom control element. The code for the custom control is shown in Section 29.4.3. This control's `Visibility` property is initially set to `Collapsed`, so it is not visible when the page loads. The `Visibility` of a control defines whether it is rendered on the screen. We also set the `Grid.RowSpan` property to 3. By taking up two rows, the GUI is blocked when the custom control is displayed, so the user can no longer interact with the main page until the control is closed. Notice that `WeatherDetailsView` is in the namespace `Weather`. This namespace (defined in line 3 of the XAML file) allows you to use the custom control in the application. The custom control must be referenced through the namespace since it is not a pre-defined control. If we did not define the namespace, there would be no way to reference `WeatherDetailsView`.

29.4.2 Obtaining and Displaying Weather Forecast Data

The **WeatherViewer** example uses Silverlight's web services, LINQ to XML and data-binding capabilities. The application's code-behind file appears in Fig. 29.4. You must insert your WeatherBug API key in line 18 in place of `"YOUR API KEY HERE"`.

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
1  // Fig. 29.4: MainPage.xaml.cs
2  // WeatherViewer displays day-by-day weather data (code-behind).
3  using System;
4  using System.Linq;
5  using System.Net;
6  using System.Text.RegularExpressions;
7  using System.Windows;
8  using System.Windows.Controls;
9  using System.Windows.Input;
10 using System.Xml.Linq;
11
12 namespace WeatherViewer
13 {
14    public partial class MainPage : UserControl
15    {
16       // object to invoke weather forecast web service
17       private WebClient weatherService = new WebClient();
18       private const string APIKey = "YOUR API KEY HERE";
19       private const int messageRowHeight = 35;
20
21       // constructor
22       public MainPage()
23       {
24          InitializeComponent();
25
26          weatherService.DownloadStringCompleted +=
27             new DownloadStringCompletedEventHandler(
28                weatherService_DownloadStringCompleted );
29       } // end constructor
30
31       // process getWeatherButton's Click event
32       private void getWeatherButton_Click(
33          object sender, RoutedEventArgs e )
34       {
35          // make sure the input string contains a five-digit number
36          if ( Regex.IsMatch( inputTextBox.Text, @"\d{5}" ) )
37          {
38             string zipcode =
39                Regex.Match( inputTextBox.Text, @"\d{5}" ).ToString();
40
41             // URL to pass to the WebClient to get our weather, complete
42             // with API key. OutputType=1 specifies XML data is needed.
43             string forecastURL =
44                "http://" + APIKey + ".api.wxbug.net/" +
45                "getForecastRSS.aspx?ACode=" + APIKey +
46                "&ZipCode=" + zipcode + "&OutputType=1";
47
48             // asynchronously invoke the web service
49             weatherService.DownloadStringAsync( new Uri( forecastURL ) );
50
51             // set the cursor to the wait symbol
52             this.Cursor = Cursors.Wait;
53          } // end if
```

Fig. 29.4 | **WeatherViewer** displays day-by-day weather data (code-behind). (Part 1 of 4.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
54            else // if input string does not contain a five-digit number,
55            {    // output an error message and do nothing else
56               messageBlock.Text = "Please enter a valid zipcode.";
57               messageRow.Height = new GridLength( messageRowHeight );
58               messageBorder.Width = forecastList.ActualWidth;
59            } // end else
60         } // end method getWeatherButton_Click
61
62         // when download is complete for web service result
63         private void weatherService_DownloadStringCompleted( object sender,
64            DownloadStringCompletedEventArgs e )
65         {
66            if ( e.Error == null )
67               DisplayWeatherForecast( e.Result );
68
69            this.Cursor = Cursors.Arrow; // arrow cursor
70         } // end method weatherService_DownloadStringCompleted
71
72         // display the received weather data
73         private void DisplayWeatherForecast( string xmlData )
74         {
75            // parse the XML data for use with LINQ
76            XDocument weatherXML = XDocument.Parse( xmlData );
77
78            XNamespace weatherNamespace =
79               XNamespace.Get( "http://www.aws.com/aws" );
80
81            // find out if the data describes the same zipcode the user
82            // entered and get the location information via LINQ to XML
83            var locationInformation =
84               from item in weatherXML.Descendants(
85                  weatherNamespace + "location" )
86               select item;
87
88            string xmlZip = string.Empty;
89            string xmlCity = string.Empty;
90            string xmlState = string.Empty;
91
92            foreach ( var item in locationInformation )
93            {
94               xmlZip = item.Element( weatherNamespace + "zip" ).Value;
95               xmlCity = item.Element( weatherNamespace + "city" ).Value;
96               xmlState = item.Element( weatherNamespace + "state" ).Value;
97            } // end for
98
99            // if the zipcodes don't match, display the data anyway,
100           // but display a message informing them of it
101           if ( !xmlZip.Equals( inputTextBox.Text ) )
102           {
103              messageBlock.Text = "Zipcode not valid; " +
104                 "displaying data for closest valid match: " +
105                 xmlCity + ", " + xmlState + ", " + xmlZip;
106              messageRow.Height = new GridLength( messageRowHeight );
```

Fig. 29.4 | **WeatherViewer** displays day-by-day weather data (code-behind). (Part 2 of 4.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
107             messageBorder.Width = forecastList.ActualWidth;
108          } // end if
109
110          // store all the data into WeatherData objects
111          var weatherInformation =
112             from item in weatherXML.Descendants(
113                weatherNamespace + "forecast" )
114             select new WeatherData
115             {
116                DayOfWeek =
117                   item.Element( weatherNamespace + "title" ).Value,
118                WeatherImage = "http://img.weather.weatherbug.com/" +
119                   "forecast/icons/localized/65x55/en/trans/" +
120                   ( ( item.Element( weatherNamespace + "image" ).Value).
121                      Substring( 51 ).Replace( ".gif", ".png" ) ),
122                MaxTemperatureF =
123                   item.Element( weatherNamespace + "high" ).Value,
124                MaxTemperatureC = convertToCelsius(
125                   item.Element( weatherNamespace + "high" ).Value),
126                MinTemperatureF =
127                   item.Element( weatherNamespace + "low" ).Value,
128                MinTemperatureC = convertToCelsius(
129                   item.Element( weatherNamespace + "low" ).Value),
130                Description =
131                   item.Element( weatherNamespace + "prediction" ).Value
132             }; // end LINQ to XML that creates WeatherData objects
133
134          // bind forecastList.ItemSource to the weatherInformation
135          forecastList.ItemsSource = weatherInformation;
136       } // end method DisplayWeatherForecast
137
138       // convert the temperature into Celsius if it's a number;
139       // cast as Integer to avoid long decimal values
140       string convertToCelsius( string fahrenheit )
141       {
142          if ( fahrenheit != "--" )
143             return ( ( Int32.Parse( fahrenheit ) - 32) *
144                5 / 9 ).ToString();
145
146          return fahrenheit;
147       } // end methodconvertToCelsius
148
149       // show details of the selected day
150       private void forecastList_SelectionChanged(
151          object sender, SelectionChangedEventArgs e )
152       {
153          // specify the WeatherData object containing the details
154          if ( forecastList.SelectedItem != null )
155             completeDetails.DataContext = forecastList.SelectedItem;
156
157          // show the complete weather details
158          completeDetails.Visibility = Visibility.Visible;
159       } // end method forecastList_SelectionChanged
```

Fig. 29.4 | **WeatherViewer** displays day-by-day weather data (code-behind). (Part 3 of 4.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
160
161        // remove displayed weather information when input zip code changes
162        private void inputTextBox_TextChanged( object sender,
163           TextChangedEventArgs e )
164        {
165           forecastList.ItemsSource = null;
166
167           // also clear the message by getting rid of its row
168           messageRow.Height = new System.Windows.GridLength( 0 );
169        } // end method inputTextBox_TextChanged
170     } // end class MainPage
171  } // end namespace WeatherViewer
```

Fig. 29.4 | **WeatherViewer** displays day-by-day weather data (code-behind). (Part 4 of 4.)

The code for the main page of the **WeatherViewer** invokes the WeatherBug web service and binds all the necessary data to the proper elements of the page. Notice that we imported the `System.Xml.Linq` namespace (line 10), which enables the LINQ to XML that is used in the example. You must also add a reference to the `System.Xml.Linq` assembly to the `WeatherViewer` Silverlight project. To do so, right click the **WeatherViewer** project in the **Solution Explorer** and select **Add Reference...**. In the dialog that appears, locate the assembly `System.Xml.Linq` in the **.NET** tab and click **OK**.

This application also uses the class `WeatherData` (line 114) that includes all the necessary weather information for a single day of the week. We created this class for you. It contains six weather information properties—`DayOfWeek`, `WeatherImage`, `MaxTemperatureF`, `MinTemperatureF`, `MaxTemperatureC`, `MinTemperatureC` and `Description`. To add the code for this class to the project, right click the **WeatherViewer** project in the **Solution Explorer** and select **Add > Existing Item...**. Find the file `WeatherData.cs` in this chapter's examples folder and click **OK**. We use this class to bind the necessary information to the `ListBox` and the custom control in our application.

Using the `WebClient` Class to Invoke a Web Service

The application's method for handling the `getWeatherButton` click grabs the zip code entered by the user in the `TextBox` and checks it against a regular-expression pattern to make sure it contains a five-digit number (line 36). If so, we store the five-digit number (lines 38–39). Next, we format the web service URL with the zip code (lines 43–46) and asynchronously invoke the web service (line 49). We use the `WebClient` class to use the web service and retrieve the desired information. We registered the event handler that handles the response in 26–28.

Line 49 calls the `weatherService` object's `DownloadStringAsync` method to invoke the web service. The web service's location must be specified as an object of class `Uri`. Class `Uri`'s constructor receives a `String` representing a uniform resource identifier, such as `"http://www.deitel.com"`. In this example, the web service is invoked asynchronously. When the web service returns its result, the `WebClient` object raises the `DownloadStringCompleted` event. Its event handler (lines 63–70) has a parameter `e` of type `DownloadStringCompletedEventArgs` which contains information returned by the web service. We

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

can use this variable's properties to get the returned XML (`e.Result`) and any errors that may have occurred during the process (`e.Error`).

Using LINQ to XML to Process the Weather Data

Once the `WebClient` has received the response, the application checks for an error (line 66). If there is no error, the application calls the `DisplayWeatherForecast` method (defined in lines 73–136).The XML that the service returns contains information about the location the user specified, which can be used for error-checking. If the user enters an incorrect zip code, the service will simply provide data for the correct zip code which is the closest match to the one the user entered. A sample of the web service's XML response appears in Fig. 29.5. The web service returns XML data that describes the high and low temperatures for the corresponding city over a period of several days. The data for each day also contains a link to an image that represents the weather for that day and a brief text description of the weather.

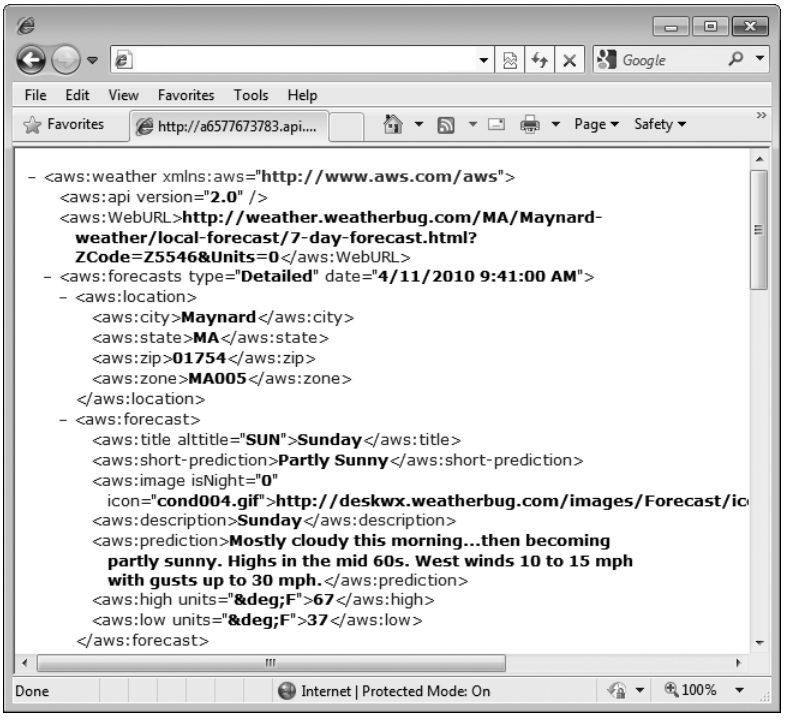


Fig. 29.5 | Sample web service XML response.

We use class `XDocument`'s **`Parse`** method (line 76) to convert a `string`—containing the contents of the XML response—to an `XDocument` to use in the LINQ to XML queries (lines 83–86 and 111–132). Lines 78–79 get the namespace for the XML returned by the web service. Each XML element in the response must be qualified with that name.

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

Error-Prevention Tip 29.1

When invoking a web service that returns XML, ensure that the namespace you specify in your code precisely matches the namespace specified in the returned XML. Otherwise, the elements in the returned XML will not be recognized in your code.

The first query pulls information from the document about the location the XML describes, which can be compared against the input string (line 101) to determine whether they denote the same location; if not, the input string is not a valid zip code, so we display an error message. We still display the data that is returned. The second query gathers the weather information and sets the corresponding values for a `WeatherData` object. The query gathers more information from the XML than is initially displayed on the main page of the application. This is because the selected object is also passed to the custom control where more detailed information about the weather is displayed. Also, the returned XML data only provides temperatures in Fahrenheit—getting them in Celsius would require a second invocation of the service with different parameters. As such, the program has a `convertToCelsius` method (lines 140–147) which converts a Fahrenheit temperature to Celsius as long as the temperature is numerical rather than two dashes `"--"` (the default when a temperature is not returned). If this is the case, `convertToCelsius` does nothing. The returned XML does not always provide both a maximum and minimum temperature for one day.

Using Data Binding to Display the Weather Data

We bind the results of the `weatherInformation` LINQ query (an `IEnumerable<T>` containing `WeatherData` objects) to the `ListBox` (line 135). This displays the summary of the weather forecast. When the user selects a particular day, we bind the `WeatherData` object for the selected day to the custom control, which displays the details for that day. The `ListBox`'s `SelectionChanged` event handler (lines 150–159) sets the `DataContext` of our custom control (line 155) to the `WeatherData` object for the selected day. The method also changes the custom control's `Visibility` to `Visible`, so the user can see the weather details.

29.4.3 Custom Controls

There are many ways to customize controls in Silverlight, including WPF's `Styles` and `ControlTemplates`. As with WPF, if deeper customization is desired, you can create **custom controls** by using the `UserControl` element as a template. The **WeatherViewer** example creates a custom control that displays detailed weather information for a particular day of the week. The control has a simple GUI and is displayed when you change your selection in the `ListBox` on the main page.

To add a new `UserControl` to the project, right click the project in the **Solution Explorer** and select **Add > New Item...**. Select the **Silverlight User Control** template and name the file `WeatherDetailsView` (Fig. 29.6).

Once added to the project, the `UserControl` can be coded similar to any other Silverlight application. The XAML code for the custom control's GUI appears in Fig. 29.7. This control contains two `StackPanels` embedded in a `Grid`. Since the aquamarine `Rectangle` (lines 12–13) in the background has an `Opacity` of 0.8, you can see that the control is treated as another element "on top of" the main page. Figure 29.8 shows the code-behind file for this control. The `Button`'s `Click` event handler collapses the control, so the user can continue interacting with the main page of the application.

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

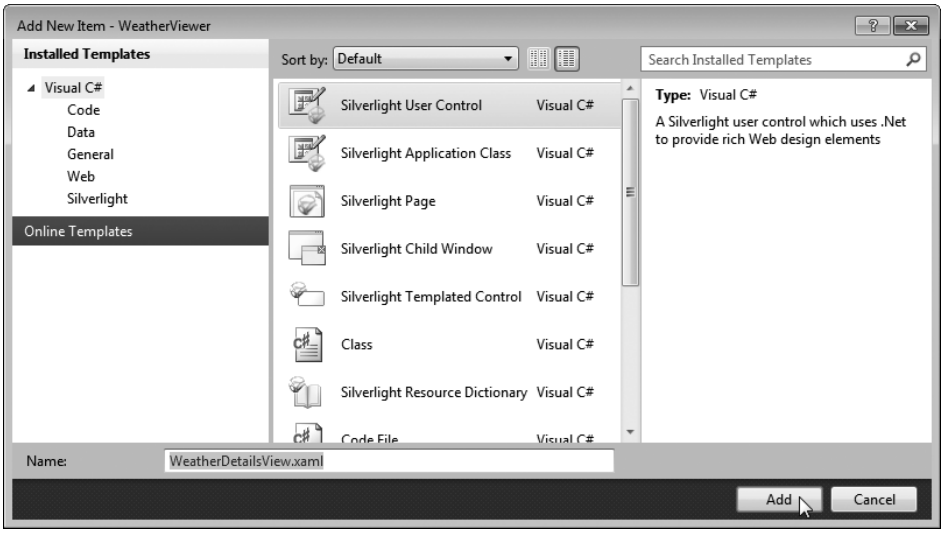


Fig. 29.6 | Adding a new `UserControl` to a Silverlight application.

```
1  <!-- Fig. 29.7: WeatherDetailsView.xaml -->
2  <!-- WeatherViewer's WeatherDetailsView custom control (XAML). -->
3  <UserControl x:Class="WeatherViewer.WeatherDetailsView"
4     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
5     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
6     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
7     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
8     mc:Ignorable="d">
9
10    <Grid x:Name="LayoutRoot" Background="White">
11       <!-- Background semitransparent rectangle -->
12       <Rectangle HorizontalAlignment="Stretch" Fill="Aquamarine"
13          VerticalAlignment="Stretch" Opacity="0.8" />
14
15       <!-- Border containing all the elements of the control -->
16       <Border CornerRadius="20" Background="AliceBlue"
17          BorderBrush="Blue" BorderThickness="4"
18          Width="400" MinHeight="175" MaxHeight="250">
19
20          <!-- StackPanel contains all the displayed weather info -->
21          <StackPanel>
22             <!-- The day's weather image -->
23             <Image Source="{Binding WeatherImage}" Margin="5" Width="55"
24                Height="58" />
25             <!-- Day of the week -->
26             <TextBlock Text="{Binding DayOfWeek}" Margin="5"
27                TextAlignment="Center" FontSize="12"
28                TextWrapping="Wrap" />
29             <!-- Displays the temperature info in C and F -->
```

Fig. 29.7 | **WeatherViewer**'s `WeatherDetailsView` custom control (XAML). (Part 1 of 2.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
30              <StackPanel HorizontalAlignment="Center"
31                 Orientation="Horizontal">
32                 <TextBlock Text="Max F:" Margin="5" FontSize="16" />
33                 <TextBlock Text="{Binding MaxTemperatureF}"
34                    Margin="5" FontSize="16" FontWeight="Bold" />
35                 <TextBlock Text="Min F:" Margin="5" FontSize="16" />
36                 <TextBlock Text="{Binding MinTemperatureF}"
37                    Margin="5" FontSize="16" FontWeight="Bold" />
38                 <TextBlock Text="Max C:" Margin="5" FontSize="16" />
39                 <TextBlock Text="{Binding MaxTemperatureC}"
40                    Margin="5" FontSize="16" FontWeight="Bold" />
41                 <TextBlock Text="Min C:" Margin="5" FontSize="16" />
42                 <TextBlock Text="{Binding MinTemperatureC}"
43                    Margin="5" FontSize="16" FontWeight="Bold" />
44              </StackPanel>
45              <!-- A description of the day's predicted weather -->
46              <TextBlock Text="{Binding Description}" FontSize="10"
47                 HorizontalAlignment="Center" MaxWidth="300"
48                 TextWrapping="Wrap" Margin="5"/>
49
50              <!-- Closes the control to go back to the main page -->
51              <Button x:Name="closeButton" Content="Close" Width="80"
52                 Click="closeButton_Click" />
53           </StackPanel>
54        </Border>
55     </Grid>
56  </UserControl>
```

Fig. 29.7 | **WeatherViewer**'s `WeatherDetailsView` custom control (XAML). (Part 2 of 2.)

```
1  // Fig. 29.8: WeatherDetailsView.xaml.cs
2  // WeatherViewer's WeatherDetailsView custom control (code-behind).
3  using System.Windows;
4  using System.Windows.Controls;
5
```

Fig. 29.8 | **WeatherViewer**'s `WeatherDetailsView` custom control (code-behind). (Part 1 of 2.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
namespace WeatherViewer
{
   public partial class WeatherDetailsView : UserControl
   {
      // constructor
      public WeatherDetailsView()
      {
         InitializeComponent();
      } // end constructor

      // close the details view
      private void closeButton_Click( object sender, RoutedEventArgs e )
      {
         this.Visibility = Visibility.Collapsed;
      } // end method closeButton_Click
   } // end class WeatherDetailsView
} // end namespace WeatherViewer
```

Fig. 29.8 | **WeatherViewer**'s `WeatherDetailsView` custom control (code-behind). (Part 2 of 2.)

29.5 Animations and the FlickrViewer

Animations in Silverlight are defined in `Storyboard`s, which are created as `Resources` of a layout control and contain one or more animation elements. When a `Storyboard`'s `Begin` method is called, its animations are applied. Silverlight has several animation types, including `DoubleAnimations`, `PointAnimations`, and `ColorAnimations`.

FlickrViewer Example

Our **FlickrViewer** example (a sample screen capture is shown in Fig. 29.9) uses a web service provided by the public photo-sharing site Flickr. The application allows you to search by tag for photos that users worldwide have uploaded to Flickr. **Tagging**—or labeling content—is part of the collaborative nature of social networking. A **tag** is any user-generated word or phrase that helps organize web content. Tagging items with self-chosen words or phrases creates a strong identification of the content. Flickr uses tags on uploaded files to improve its photo-search service, giving the user better results. To run this example on your computer, *you need to obtain your own Flickr API key at* `www.flickr.com/services/api/keys/` *and add it to the* `MainPage.xaml.cs` *file* (which we discuss shortly). This key is a unique string of characters and numbers that enables Flickr to track usage of their APIs.

The application shows you thumbnails of the first 20 (or fewer if there are not 20) public results (as specified in the URL that invokes the web service) and allows you to click a thumbnail to view its full-sized image. As you change your selection, the application animates out the previously selected image and animates in the new selection. The `Border` shrinks until the current `Image` is no longer visible, then expands to display the new selected `Image`.

As shown in Fig. 29.9, you can type one or more tags (e.g., "`deitel flowers`") into the application's `TextBox`. When you click the **Search** `Button`, the application invokes the Flickr web service, which responds with an XML document containing links to the photos that match the tags. The application parses the XML and displays thumbnails of these photos. The application's XAML is shown in Fig. 29.10.

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

Fig. 29.9 | **FlickrViewer** allows users to search photos by tag.

```
1  <!-- Fig. 29.10: MainPage.xaml -->
2  <!-- FlickrViewer allows users to search for tagged photos (XAML). -->
3  <UserControl x:Class="FlickrViewer.MainPage"
4     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
5     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
6     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
7     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
8     mc:Ignorable="d">
9
10    <Grid x:Name="LayoutRoot" Background="White">
11       <Grid.RowDefinitions>
12          <!-- Defines the rows of the main grid -->
13          <RowDefinition Height="Auto" />
14          <RowDefinition x:Name="imageRow" />
15          <RowDefinition Height="Auto" />
16       </Grid.RowDefinitions>
17
18       <Grid.Resources> <!-- Contains the page's animations -->
19
20          <!-- Enlarges the Border to display a new image -->
21          <Storyboard x:Name="animateIn"
22             Storyboard.TargetName="largeCoverImage"
23             Completed="animateIn_Completed">
24             <DoubleAnimation x:Name="animate"
25                Storyboard.TargetProperty="Height" Duration="0:0:0.75" >
```

Fig. 29.10 | **FlickrViewer** allows users to search for tagged photos (XAML). (Part 1 of 3.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
26                <DoubleAnimation.EasingFunction>
27                   <ElasticEase Springiness="10"/>
28                </DoubleAnimation.EasingFunction>
29             </DoubleAnimation>
30          </Storyboard>
31
32          <!-- Collapses the Border in preparation for a new image -->
33          <Storyboard x:Name="animateOut"
34             Storyboard.TargetName="largeCoverImage"
35             Completed="animateOut_Completed">
36             <DoubleAnimation Storyboard.TargetProperty="Height" To="60"
37                Duration="0:0:0.25" />
38          </Storyboard>
39
40          <!-- Rotates the Search button in three dimensions -->
41          <Storyboard x:Name="buttonRotate"
42             Storyboard.TargetName="buttonProjection">
43             <DoubleAnimation x:Name="rotX"
44                Storyboard.TargetProperty="RotationX" Duration="0:0:0.5" />
45             <DoubleAnimation x:Name="rotY"
46                Storyboard.TargetProperty="RotationY" Duration="0:0:0.5" />
47             <DoubleAnimation x:Name="rotZ"
48                Storyboard.TargetProperty="RotationZ" Duration="0:0:0.5" />
49          </Storyboard>
50       </Grid.Resources>
51
52       <!-- Contains the search box and button for user interaction -->
53       <StackPanel Grid.Row="0" Orientation="Horizontal">
54          <TextBox x:Name="searchBox" Width="150" />
55          <Button x:Name="searchButton" Content="Search" Width="75"
56             Click="searchButton_Click">
57             <!-- We must declare and name the button's projection in
58                order to rotate it -->
59             <Button.Projection>
60                <PlaneProjection x:Name="buttonProjection" />
61             </Button.Projection>
62          </Button>
63       </StackPanel>
64
65       <!-- Border that contains the large main image -->
66       <Border Grid.Row="1" x:Name="largeCoverImage" Height="60"
67          BorderBrush="Black" BorderThickness="10" CornerRadius="10"
68          Padding="20" Margin="10" HorizontalAlignment="Center">
69          <Border.Background>
70             <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
71                <GradientStop Offset="0" Color="Black" />
72                <GradientStop Offset="1" Color="LightGray" />
73             </LinearGradientBrush>
74          </Border.Background>
75
76          <!-- Displays the image that the user selected -->
77          <Image Source="{Binding}" Stretch="Uniform" />
78       </Border>
```

Fig. 29.10 | **FlickrViewer** allows users to search for tagged photos (XAML). (Part 2 of 3.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
79
80          <!-- Listbox displays thumbnails of the search results -->
81          <ListBox x:Name="thumbsListBox" Grid.Row="2"
82             HorizontalAlignment="Center"
83             SelectionChanged="thumbsListBox_SelectionChanged">
84             <ListBox.ItemsPanel>
85                <ItemsPanelTemplate>
86                   <StackPanel Orientation="Horizontal"/>
87                </ItemsPanelTemplate>
88             </ListBox.ItemsPanel>
89
90             <ListBox.ItemTemplate>
91                <DataTemplate>
92                   <Image Source="{Binding}" Margin="10" />
93                </DataTemplate>
94             </ListBox.ItemTemplate>
95          </ListBox>
96       </Grid>
97    </UserControl>
```

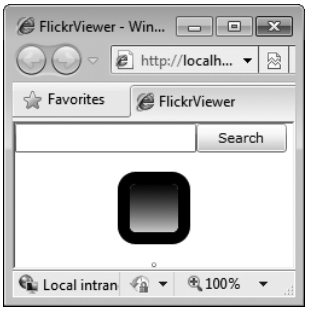


Fig. 29.10 | **FlickrViewer** allows users to search for tagged photos (XAML). (Part 3 of 3.)

Lines 18–50 define the `Grid`'s `Resources`, which contain three `Storyboard` elements to facilitate various animations. The `animateIn` `Storyboard` (lines 21–30) contains a `DoubleAnimation` that animates the `Height` property of the `largeCoverImage`'s `Border`. Though this animation is a `From`/`To`/`By` animation, the `To` property is not set. We set this value in the C# code to allow the border to fill the available space in the window regardless of the browser window size. Lines 26–28 contain an **`EasingFunction`**. Silverlight animations provide smooth, linear change of an item's attribute over a set period of time, but an `EasingFunction` allows animations to follow other patterns. This particular `EasingFunction`, `ElasticEase`, will cause `animateIn` to oscillate like it is attached to a spring. The `animateOut` `Storyboard` (lines 33–38) shrinks the `Border` until the image inside is no longer visible. `Storyboards` can also contain multiple animations; the `buttonRotate` `Storyboard` (lines 41–49) will rotate the button in all three dimensions when it's clicked, but a separate `DoubleAnimation` must be declared for each dimension. Like `animateIn`, `buttonRotate`'s `To` property is set programmatically.

The rest of the layout is similar to that of the **WeatherViewer**. Lines 11–16 define the main `Grid`'s three rows. The first row contains a `StackPanel` with an embedded search `TextBox` and a `Button` (lines 53–63). To give the three-dimensional `buttonRotate` anima-

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

tion a target, the `Button`'s `Projection` must be declared and named; an item's `Projection` lets you control its rotation in three dimensions. The second row contains the `Border` with an embedded `Image` (lines 66–78) to display the large version of the selected thumbnail. The third row contains the `ListBox` (lines 81–95), which displays the thumbnails of the photos returned from Flickr. This `ListBox` is organized and coded in the same way as in the **WeatherViewer**, except that the `DataTemplate` contains only one `Image`—one of the photos returned by the web service. The screen capture in Fig. 29.10 shows the empty layout of the **FlickrViewer** before the user enters a search query.

The C# code for the application can be seen in Fig. 29.11. This example uses web services and LINQ to XML.

```
1  // Fig. 29.11: MainPage.xaml.cs
2  // FlickrViewer allows users to search for photos (code-behind).
3  using System;
4  using System.Linq;
5  using System.Net;
6  using System.Net.NetworkInformation;
7  using System.Windows;
8  using System.Windows.Controls;
9  using System.Xml.Linq;
10
11 namespace FlickrViewer
12 {
13    public partial class MainPage : UserControl
14    {
15       // Flickr API key
16       private const string KEY = "YOUR API KEY HERE";
17
18       // object used to invoke Flickr web service
19       private WebClient flickr = new WebClient();
20
21       // constructor
22       public MainPage()
23       {
24          InitializeComponent();
25          flickr.DownloadStringCompleted +=
26             new DownloadStringCompletedEventHandler(
27                flickr_DownloadStringCompleted );
28       } // end constructor
29
30       // when the photo selection has changed
31       private void thumbsListBox_SelectionChanged( object sender,
32          SelectionChangedEventArgs e )
33       {
34          // set the height back to a value so that it can be animated
35          largeCoverImage.Height = largeCoverImage.ActualHeight;
36
37          animateOut.Begin(); // begin shrinking animation
38       } // end method thumbsListBox_SelectionChanged
39
```

Fig. 29.11 | **FlickrViewer** allows users to search for tagged photos (code-behind). (Part 1 of 3.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
40         // this makes sure that the border will resize with the window
41         private void animateIn_Completed( object sender, EventArgs e )
42         {
43            largeCoverImage.Height = double.NaN; // image height = *
44         } // end method animateIn_Completed
45
46         // once the nested image is no longer visible
47         private void animateOut_Completed( object sender, EventArgs e )
48         {
49            if ( thumbsListBox.SelectedItem != null )
50            {
51               // grab the URL of the selected item's full image
52               string photoURL =
53                  thumbsListBox.SelectedItem.ToString().Replace(
54                  "_t.jpg", ".jpg" );
55
56               largeCoverImage.DataContext = photoURL;
57
58               animate.To = imageRow.ActualHeight - 20;
59               animateIn.Begin();
60            } // end if
61         } // end method animateOut_Completed
62
63         // begin the search when the user clicks the search button
64         private void searchButton_Click( object sender, RoutedEventArgs e )
65         {
66            // if network is available, get images
67            if ( NetworkInterface.GetIsNetworkAvailable() )
68            {
69               // Flickr's web service URL for searches
70               var flickrURL = string.Format(
71                  "http://api.flickr.com/services/rest/?" +
72                  "method=flickr.photos.search&api_key={0}&tags={1}" +
73                  "&tag_mode=all&per_page=20&privacy_filter=1", KEY,
74                  searchBox.Text.Replace( " ", "," ) );
75
76               // invoke the web service
77               flickr.DownloadStringAsync( new Uri( flickrURL ) );
78
79               // disable the search button
80               searchButton.Content = "Loading...";
81               searchButton.IsEnabled = false;
82
83               flipButton(); // start 3D Button rotation animation
84            } // end if
85            else
86               MessageBox.Show( "ERROR: Network not available!" );
87         } // end method searchButton_Click
88
89         // once we have received the XML file from Flickr
90         private void flickr_DownloadStringCompleted( object sender,
91            DownloadStringCompletedEventArgs e )
92         {
```

Fig. 29.11 | **FlickrViewer** allows users to search for tagged photos (code-behind). (Part 2 of 3.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
93            searchButton.Content = "Search";
94            searchButton.IsEnabled = true;
95
96            if ( e.Error == null )
97            {
98               // parse the data with LINQ
99               XDocument flickrXML = XDocument.Parse( e.Result );
100
101              // gather information on all photos
102              var flickrPhotos =
103                 from photo in flickrXML.Descendants( "photo" )
104                 let id = photo.Attribute( "id" ).Value
105                 let secret = photo.Attribute( "secret" ).Value
106                 let server = photo.Attribute( "server" ).Value
107                 let farm = photo.Attribute( "farm" ).Value
108                 select string.Format(
109                    "http://farm{0}.static.flickr.com/{1}/{2}_{3}_t.jpg",
110                    farm, server, id, secret);
111
112              // set thumbsListBox's item source to the URLs we received
113              thumbsListBox.ItemsSource = flickrPhotos;
114           } // end if
115        } // end method flickr_DownloadStringCompleted
116
117        // perform 3D Button rotation animation
118        void flipButton()
119        {
120           // give all the animations a new goal
121           rotX.To = buttonProjection.RotationX + 360;
122           rotY.To = buttonProjection.RotationY + 360;
123           rotZ.To = buttonProjection.RotationZ + 360;
124
125           buttonRotate.Begin(); // start the animation
126        } // end method flipButton
127     } // end class MainPage
128  } // end namespace FlickrViewer
```

Fig. 29.11 | **FlickrViewer** allows users to search for tagged photos (code-behind). (Part 3 of 3.)

The library `System.Net.NetworkInformation` contains tools to check the status of the network. Using the `NetworkInterface.GetIsNetworkAvailable` function (line 67), the program attempts to connect to Flickr only if connected to a network (lines 67–84) and simply displays an error message otherwise (lines 85–86).

Line 16 defines a constant `String` for the API key that is required to use the Flickr API. To run this application insert your Flickr API key here.

Recall that the `To` property of the `DoubleAnimation` in the `animateIn Storyboard` is set programatically. Line 58 sets the `To` property to the `Height` of the page's second row (minus 20 to account for the `Border`'s `Margin`), animating the `Height` to the largest possible value while keeping the `Border` completely visible on the page.

For animations to function properly, the properties being animated must contain numeric values—relative values `"*"` and `"Auto"` do not work. So before `animateOut` begins, we assign the value `largeImageCover.ActualHeight` to the `Border`'s `Height` (line

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

35). When the `Border` is not being animated, we want it to take up as much space as possible on screen while still being resizable based on the changing size of the browser window. Line 43 resets the `Border`'s `Height` back to `Double.NaN`, which allows the border to be resized with the window.

Notice that when you click a new picture that you have not previously viewed, the `Border`'s `Height` increases without displaying a new picture inside. This is because the animation begins before the application can download the entire image. The picture is not displayed until its download is complete. If you click the thumbnail of an image you've viewed previously, it displays properly, because the image has already been downloaded to your system and cached by the browser. Viewing the image again causes it to be loaded from the browser's cache rather than over the web.

Lines 102–110 of Fig. 29.11 use a LINQ query to gather the necessary information from the attributes of the `photo` elements in the XML returned by the web service. A sample of the XML response is shown in Fig. 29.12. The four values collected are required to form the URL to the online photos. The thumbnail URLs are created in lines 108–110 in the LINQ query's `select` clause. The "`_t`" before the "`.jpg`" in each URL indicates that we want the thumbnail of the photo rather than the full-sized file. These URLs are passed to the `ItemsSource` property of `thumbsListBox`, which displays all the thumbnails at the bottom of the page. To load the large `Image`, use the URL of the thumbnail and remove the `"_t"` from the link (lines 52–54), then change the source of the `Image` element in the `Border` (line 56). Notice that the data binding in lines 77 and 92 of Fig. 29.10 use the simple `"{Binding}"` syntax. This works because we're binding a single `String` to the object rather than an object with several properties.

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <rsp stat="ok">
3     <photos page="1" pages="1" perpage="20" total="5">
4        <photo id="2608518732" owner="8832668@N04" secret="76dab8eb42"
5           server="3185" farm="4" title="Red Flowers 1" ispublic="1"
6           isfriend="0" isfamily="0" />
7        <photo id="2608518654" owner="8832668@N04" secret="57d35c8f64"
8           server="3293" farm="4" title="Lavender Flowers" ispublic="1"
9           isfriend="0" isfamily="0" />
10       <photo id="2608518890" owner="8832668@N04" secret="98fcb5fb42"
11          server="3121" farm="4" title="Yellow Flowers" ispublic="1"
12          isfriend="0" isfamily="0" />
13       <photo id="2608518370" owner="8832668@N04" secret="0099e12778"
14          server="3076" farm="4" title="Fuscia Flowers" ispublic="1"
15          isfriend="0" isfamily="0" />
16       <photo id="2607687273" owner="8832668@N04" secret="4b630e31ba"
17          server="3283" farm="4" title="Red Flowers 2" ispublic="1"
18          isfriend="0" isfamily="0" />
19    </photos>
20 </rsp>
```

Fig. 29.12 | Sample XML response from the Flickr APIs.

Method `flipButton` (lines 118–126) activates the `buttonRotate Storyboard`. The method sets the `To` property in all three dimensions to 360 degrees greater than its current

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

value. When we call the `Storyboard`'s `Begin` method, the button rotates 360 degrees in each dimension.

Out-of-Browser Experience

Silverlight's **out-of-browser experiences** enable you to configure a Silverlight application so that any user can download a local copy of it and place a shortcut to it on their desktop and in their **Start** menu. To configure the FlickerViewer application for an out-of-browser experience, perform the following steps:

1. Right click the **FlickerViewer** project in the **Solution Explorer** and select **Properties**.
2. Ensure that **Enable running application out of the browser** is checked.
3. Click the **Out-of-Browser Settings...** button.
4. In the **Out-of-Browser Settings** dialog (Fig. 29.13), you can configure the application's settings, including the window's title, width and height. You can also specify the shortcut name, the application's description and icons to represent your application. In this case, we kept the default settings, but set the width and height of the window.
5. Click **OK** to save your settings.

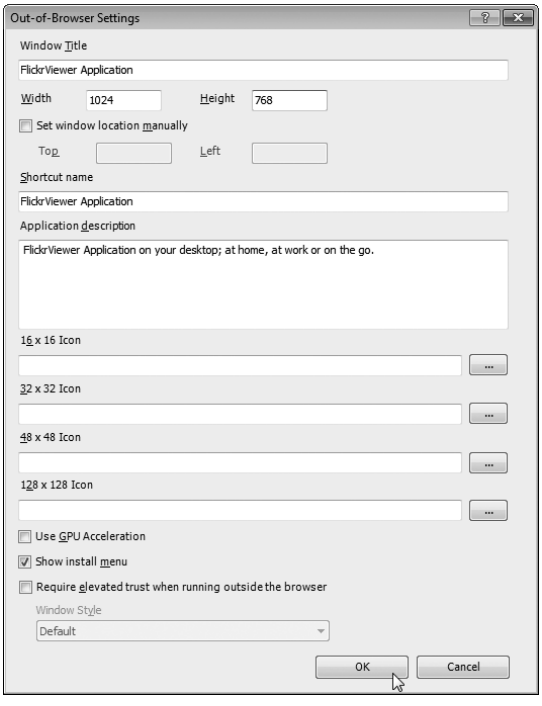


Fig. 29.13 | **Out-of-Browser Settings** dialog.

Once you've configured the application for an out-of-browser experience, the user can right click the application in the browser to see the menu in Fig. 29.14. Selecting **Install**

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

FlickrViewer onto this computer... presents you with a dialog that allows you to choose where you want the shortcut for the application to be installed. After clicking **OK**, the application will execute in its own window. In the future, you can run the Silverlight application from its shortcut.

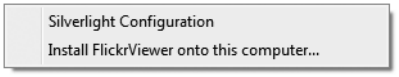


Fig. 29.14 | Right-click menu for a Silverlight application that supports an out-of-browser experience.

29.6 Images and Deep Zoom

One feature in Silverlight that is not in WPF is the **`MultiScaleImage`**. In most desktop applications, you'll have no trouble viewing and zooming in on a high-resolution image. Doing this over the Internet is problematic, however, because transferring large images usually takes significant time, which prevents web-based applications from having the feel of desktop applications.

This problem is addressed by Silverlight's **deep zoom** capabilities, which use `MultiScaleImages` to allow you to zoom far into an image in a web browser while maintaining quality. One of the best demonstrations of this technology is the Hard Rock Cafe's memorabilia page (`memo.hardrock.com`), which uses Silverlight's deep zoom capabilities to display a large collage of rock and roll memorabilia. You can zoom in on any individual item to see its high-resolution image. The photographs were taken at such high resolution that you can actually see fingerprints on the surfaces of some of the guitars!

Deep zoom works by sending only the necessary image data for the part of the image you are viewing to your machine. To split an image or collage of images into the Silverlight-ready format used by `MultiScaleImages`, you use the **Deep Zoom Composer** (available from `www.microsoft.com/uk/wave/software-deepzoom.aspx`). The original images are split into smaller pieces to support various zoom levels. This enables the server to send smaller chunks of the picture rather than the entire file. If you zoom in close to an image, the server sends only the small section that you are viewing at its highest available resolution (which depends on the resolution of the original image). If you zoom out, the server sends only a lower-resolution version of the image. In either case, the server sends just enough data to give the user a rich image-viewing experience.

A `MultiScaleImage`'s `Source` is an XML document—created by Deep Zoom Composer. The `MultiScaleImage` uses the data in the XML to display an image or collage of images. A **`MultiScaleSubImage`** of a `MultiScaleImage` contains information on a single image in a collage.

The DeepZoomCoverCollage Example

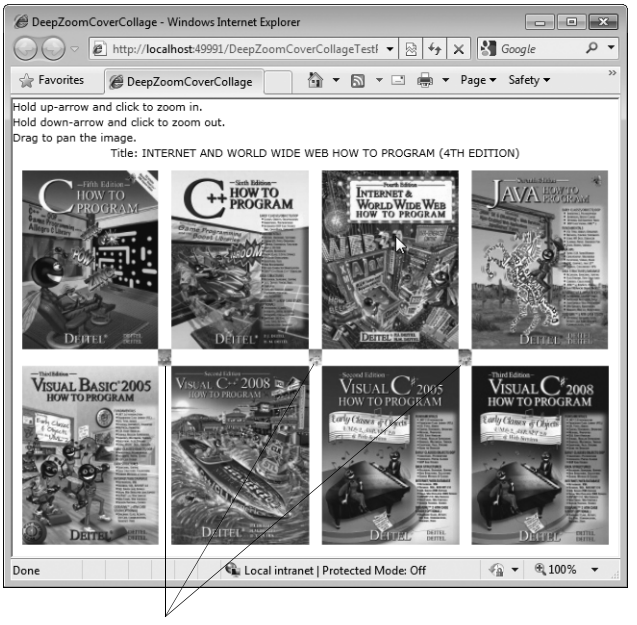
Our **DeepZoomCoverCollage** application contains a high-resolution collage of 12 of our book covers. You can zoom in and out and pan the image with simple keystroke and mouse-click combinations. Figure 29.15 shows screen captures of the application.

Figure 29.15(a) shows the application when it's first loaded with all 12 cover images displayed. Eight large images and three tiny images are clearly visible. One cover is hidden

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

within one of these eleven covers. Test-run the program to see if you can find it. Figure 29.15(b) shows the application after we've zoomed in closely on the leftmost small

a)



Small images nested among larger images in the collage

b)

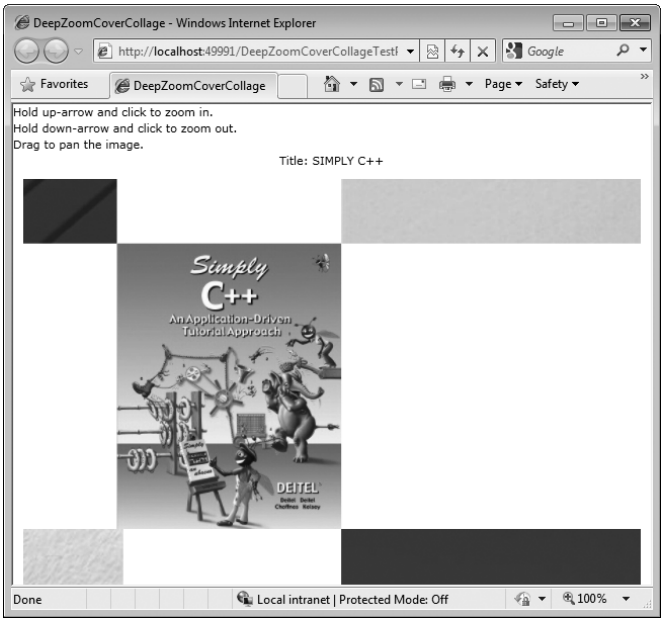


Fig. 29.15 | Main page of the **DeepZoomCoverCollage**. (Part 1 of 2.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

c)

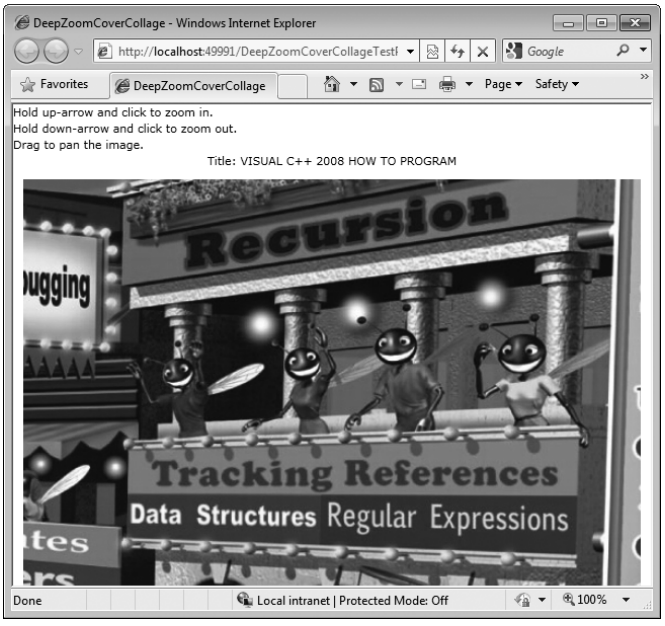


Fig. 29.15 | Main page of the **DeepZoomCoverCollage**. (Part 2 of 2.)

cover image. As you can see in the second screen capture, the small cover image still comes up clearly, because it was originally created in the Deep Zoom Composer with a high-resolution image. Figure 29.15(c) shows the application with an even deeper zoom on a different cover. Rather than being pixelated, the image displays the details of the original picture.

29.6.1 Getting Started With Deep Zoom Composer

To create the collection of files that is used by `MultiScaleImage`, you need to import the image or set of images into Deep Zoom Composer. When you first open the program, create a new project through the **File** menu, and specify the project's **Name** and **Location**. We named the project **CoverCollage**. Figure 29.16 shows the **New Project** dialog.

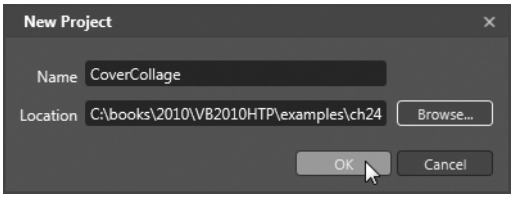


Fig. 29.16 | Deep Zoom Composer's **New Project** dialog.

The **Import** tab in Deep Zoom Composer is displayed by default. It enables you to add the image(s) that you want in the collage. Click the **Add Image...** button to add your images. (We provided our book-cover images with this chapter's examples in the `Cover`

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

`Images` folder.) Once you've added your images, you'll see their thumbnails on the right side of the window. A larger version of the selected image appears in the middle of the window. Figure 29.17 shows the window with the **Import** tab open after the book-cover images have been imported to the project.

Fig. 29.17 | Deep Zoom Composer showing the imported image files.

For our **CoverCollage** example, we use high-resolution `.jpg` images. Deep Zoom Composer also supports `.tif`, `.bmp` and `.png` formats. After importing the images, you can go to the **Compose** tab to organize them on your collage.

Drag the thumbnail of each desired image onto the main canvas of the window. When you drag a file into the collage, its thumbnail is grayed out in the side bar and you cannot add it to the collage again. Figure 29.18 shows what the composer looks like, once you bring files into the project.

When images are in the composition, you can move the images to the canvas and resize them to be as large or small as you want. Deep Zoom Composer has features such as snapping and alignment tools that help you lay out the images. Yellow pins throughout the collage in Fig. 29.18(a) indicate that there are small images at those locations. You can zoom in on the composition by scrolling the mouse wheel to see the smaller image. Figure 29.18(b) shows the smaller cover marked by one of the pins. A small screen in the bottom-left corner shows the entire collage and a white rectangle indicating the view displayed in the window.

The panel on the right showing all the images also has a **Layer View** option, which indicates the layer ordering of all the composition's images. This view is used to control the order of overlapping images. The layers can be rearranged to allow you to place certain images on top of others.

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

a)

b)

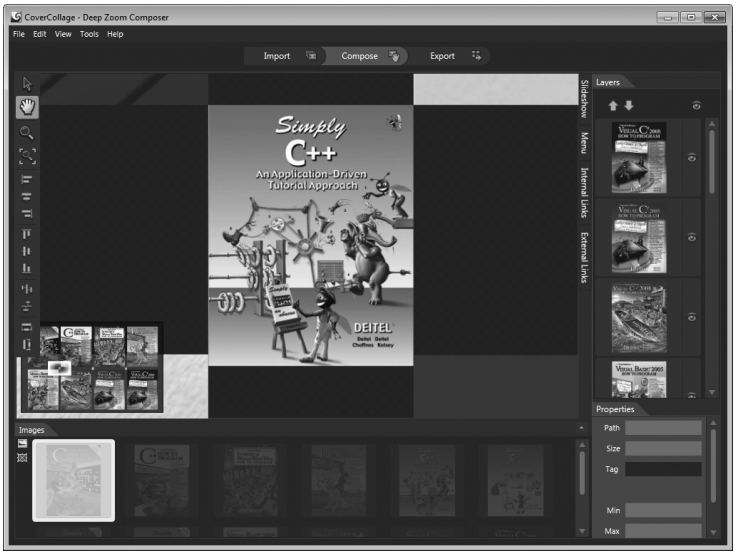


Fig. 29.18 | Deep Zoom Composer showing the editable composition.

Once you have a completed collage, go to the **Export** tab to export the files to be used by a `MultiScaleImage` in your application. Figure 29.19 shows the contents of the window when the **Export** tab is open.

You'll need to name the project. For this example, select the **Custom** tab, then name the project `CoverCollageCollection` and keep the default **Export Location**. The files are

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

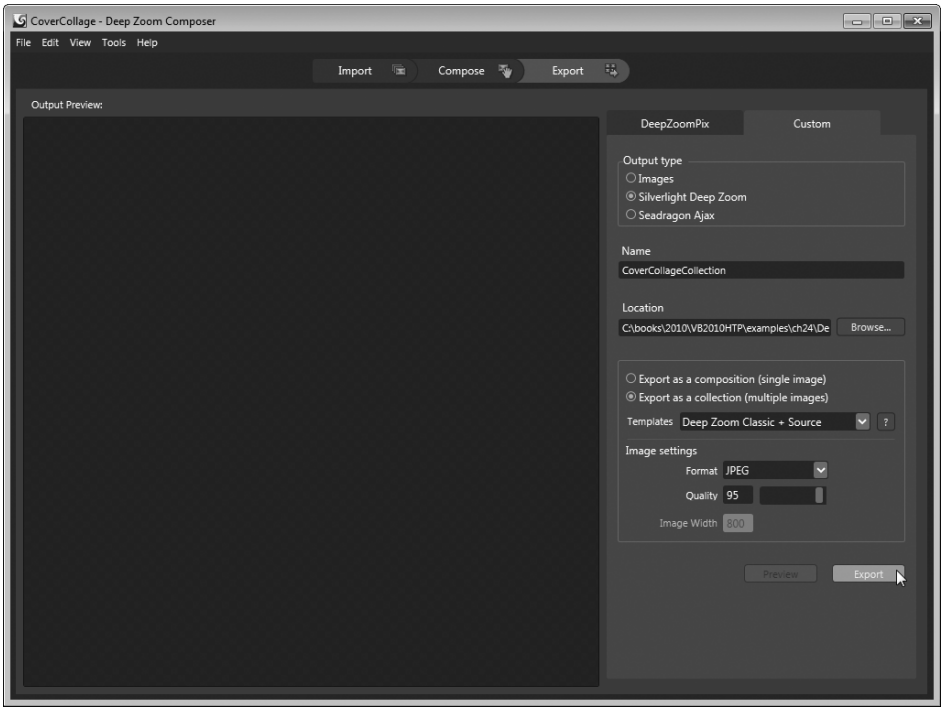


Fig. 29.19 | Deep Zoom Composer's exporting capabilities.

exported to a new folder inside the directory that you created earlier for the Deep Zoom Composer project. By default, Deep Zoom Composer selects the **Export as Collection** option using a JPEG file format. By exporting as a collection instead of a composition, subimage information is included in the output XML files. Keep the JPEG **Quality** at 95—lower values result in smaller file sizes and lower-quality images. From **Templates**, select the **Deep Zoom Classic + Source** option, then click **Export**. Once the project is done exporting, you'll be ready to import these files into a Silverlight project and use them to create a deep zoom application.

29.6.2 Creating a Silverlight Deep Zoom Application

Deep zoom images are created in Silverlight Projects by using the `MultiScaleImage` element, which takes an XML file as its source. A `MultiScaleImage` can be treated in the XAML code similar to a simple `Image` element. Previously, we showed you screen captures of the `DeepZoomCoverCollage` example. Figure 29.20 is the XAML code that produces the layout of this application.

```
1  <!-- Fig. 29.20: MainPage.xaml -->
2  <!-- DeepZoomCoverCollage employs Silverlight's deep zoom (XAML). -->
3  <UserControl x:Class="DeepZoomCoverCollage.MainPage"
```

Fig. 29.20 | **DeepZoomCoverCollage** employs Silverlight's deep zoom (XAML). (Part 1 of 2.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
4     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
5     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
6     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
7     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
8     mc:Ignorable="d" KeyDown="mainPage_KeyDown" KeyUp="mainPage_KeyUp">
9
10    <Grid x:Name="LayoutRoot" Background="White">
11
12       <!-- instructions on how to interact with the page -->
13       <StackPanel Orientation="Vertical">
14          <TextBlock Text="Hold up-arrow and click to zoom in." />
15          <TextBlock Text="Hold down-arrow and click to zoom out." />
16          <TextBlock Text="Drag to pan the image."  />
17
18          <!-- book title -->
19          <TextBlock x:Name="titleTextBlock" Text="Title:"
20             HorizontalAlignment="Center" />
21
22          <!-- deep zoom collage that was created in composer -->
23          <MultiScaleImage x:Name="Image" Margin="10"
24             Source="/GeneratedImages/dzc_output.xml"
25             MouseLeave="Image_MouseLeave" MouseMove="Image_MouseMove" />
26             MouseLeftButtonDown="Image_MouseLeftButtonDown"
27             MouseLeftButtonUp="Image_MouseLeftButtonUp" />
28       </StackPanel>
29    </Grid>
30 </UserControl>
```

Fig. 29.20 | **DeepZoomCoverCollage** employs Silverlight's deep zoom (XAML). (Part 2 of 2.)

The main page contains only a `StackPanel` with `TextBlocks` that display instructions, a `TextBlock` to display the selected book's title and the `MultiScaleImage` to display the collage we created in the previous section. To use the collage, you must add the entire `GeneratedImages` folder to your Silverlight project. If you kept the default Deep Zoom Composer export location, this folder can be found in the `CoverCollage` project's folder under the subfolder `\Exported Data\covercollagecollection\DeepZoomProject-Site\ClientBin\`. Copy the `GeneratedImages` folder into the `ClientBin` folder of the web application project by dragging it from Windows Explorer onto that folder in the **Solution Explorer**. If the `CoverCollageCollection` folder was copied correctly, you should see a `GeneratedImages` folder (Fig. 29.21). You can now refer to this collection in your application.

Once the necessary files are in the project, they can be used by the `MultiScaleImage` element that displays the deep zoom image. Line 24 of Fig. 29.20 defines the source of the `MultiScaleImage` to `"/GeneratedImages/dzc_output.xml"`. The source address in this case is relative to the **ClientBin**, meaning that the application searches for the given path in the **ClientBin** folder of the project. Now that the `MultiScaleImage` is ready, we can program the application's event handlers for zooming and panning the image, and for displaying a book's title when its cover is clicked (Fig. 29.22). We use a LINQ query to find the title of the cover image the user selects. We have several instance variables that help us determine which operation is to occur when you click the mouse.

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

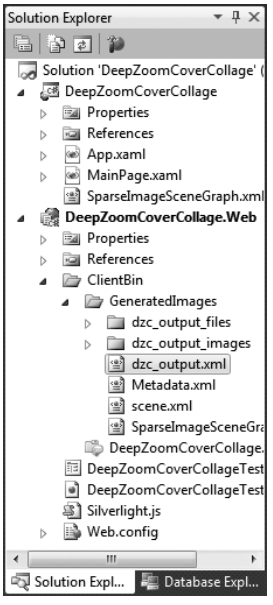


Fig. 29.21 | **Solution Explorer** after the deep zoom files have been added to the project.

```
1  // Fig. 29.22: MainPage.xaml.cs
2  // DeepZoomCoverCollage employs Silverlight's deep zoom (code-behind).
3  using System;
4  using System.IO;
5  using System.Linq;
6  using System.Windows;
7  using System.Windows.Controls;
8  using System.Windows.Input;
9  using System.Xml.Linq;
10
11 namespace DeepZoomCoverCollage
12 {
13    public partial class MainPage : UserControl
14    {
15       private const double ZOOMFACTOR = 2.0;
16
17       private bool zoomIn = false; // true if Up button is pressed
18       private bool zoomOut = false; // true if Down button is pressed
19       private bool mouseDown = false; // true if mouse button is down
20       private Point currentPosition; // position of viewport when clicked
21       private Point dragOffset; // mouse offset used for panning
22
23       // constructor
24       public DeepZoomCoverCollagePage()
25       {
```

Fig. 29.22 | **DeepZoomCoverCollage** employs Silverlight's deep zoom (code-behind). (Part 1 of 4.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
26            InitializeComponent();
27         } // end constructor
28
29         //  when a key is pressed, set the correct variables to true
30         private void mainPage_KeyDown( object sender, KeyEventArgs e )
31         {
32            if ( e.Key == Key.Up ) // button pressed is Up
33            {
34               zoomIn = true; // prepare to zoom in
35            } // end if
36            else if ( e.Key == Key.Down ) // button pressed is Down
37            {
38               zoomOut = true; // prepare to zoom out
39            } // end else if
40         } // end method mainPage_KeyDown
41
42         // when a key is released, set the correct variables to false
43         private void mainPage_KeyUp( object sender, KeyEventArgs e )
44         {
45            if ( e.Key == Key.Up ) // button released is Up
46            {
47               zoomIn = false; // don't zoom in
48            } // end if
49            else if ( e.Key == Key.Down ) // button released is Down
50            {
51               zoomOut = false; // don't zoom out
52            } // end else if
53         } // end method mainPage_KeyUp
54
55         // when the mouse leaves the area of the image we don't want to pan
56         private void Image_MouseLeave( object sender, MouseEventArgs e )
57         {
58            mouseDown = false; // if mouse leaves area, no more panning
59         } // end method Image_MouseLeave
60
61         // handle events when user clicks the mouse
62         private void Image_MouseLeftButtonDown( object sender,
63            MouseButtonEventArgs e )
64         {
65            mouseDown = true; // mouse button is down
66            currentPosition = Image.ViewportOrigin; // viewport position
67            dragOffset = e.GetPosition( Image ); // mouse position
68
69            // logical position (between 0 and 1) of mouse
70            Point click = Image.ElementToLogicalPoint( dragOffset );
71
72            if ( zoomIn ) // zoom in when Up key is pressed
73            {
74               Image.ZoomAboutLogicalPoint( ZOOMFACTOR, click.X, click.Y );
75            } // end if
```

Fig. 29.22 | **DeepZoomCoverCollage** employs Silverlight's deep zoom (code-behind). (Part 2 of 4.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
76            else if ( zoomOut ) // zoom out when Down key is pressed
77            {
78               Image.ZoomAboutLogicalPoint( 1 / ZOOMFACTOR,
79                  click.X, click.Y );
80            } // end else if
81
82            // determine which book cover was pressed to display the title
83            int index = SubImageIndex( click );
84
85            if ( index > -1 )
86            {
87               titleTextBlock.Text = string.Format(
88                  "Title: {0}", GetTitle( index ) );
89            }
90            else // user clicked a blank space
91            {
92               titleTextBlock.Text = "Title:";
93            } // end else
94         } // end method Image_MouseLeftButtonDown
95
96         // if the mouse button is released, we don't want to pan anymore
97         private void Image_MouseLeftButtonUp( object sender,
98            MouseButtonEventArgs e )
99         {
100           mouseDown = false; // no more panning
101        } // end method Image_MouseLeftButtonUp
102
103        // handle when the mouse moves: panning
104        private void Image_MouseMove( object sender, MouseEventArgs e )
105        {
106           // if no zoom occurs, we want to pan
107           if ( mouseDown && !zoomIn && !zoomOut )
108           {
109              Point click = new Point(); // records point to move to
110              click.X = currentPosition.X - Image.ViewportWidth * (
111                 e.GetPosition( Image ).X - dragOffset.X ) /
112                 Image.ActualWidth;
113              click.Y = currentPosition.Y - Image.ViewportWidth * (
114                 e.GetPosition( Image ).Y - dragOffset.Y ) /
115                 Image.ActualWidth;
116              Image.ViewportOrigin = click; // pans the image
117           } // end if
118        } // end method Image_MouseMove
119
120        // returns the index of the clicked subimage
121        private int SubImageIndex( Point click )
122        {
123           // go through images such that images on top are processed first
124           for ( int i = Image.SubImages.Count - 1; i >= 0; i-- )
125           {
```

Fig. 29.22 | **DeepZoomCoverCollage** employs Silverlight's deep zoom (code-behind). (Part 3 of 4.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
126             // select a single subimage
127             MultiScaleSubImage subImage = Image.SubImages[ i ];
128
129             // create a rect around the area of the cover
130             double scale = 1 / subImage.ViewportWidth;
131             Rect area = new Rect( -subImage.ViewportOrigin.X * scale,
132                -subImage.ViewportOrigin.Y * scale, scale, scale /
133                subImage.AspectRatio );
134
135             if ( area.Contains( click ) )
136             {
137                return i; // return the index of the clicked cover
138             } // end if
139          } // end for
140          return -1; // if no cover was clicked, return -1
141       } // end method SubImageIndex
142
143       // returns the title of the subimage with the given index
144       private string GetTitle( int index )
145       {
146          // XDocument that contains info on all subimages in the collage
147          XDocument xmlDocument =
148             XDocument.Load( "SparseImageSceneGraph.xml" );
149
150          // LINQ to XML to find the title based on index of clicked image
151          var bookTitle =
152             from info in xmlDocument.Descendants( "SceneNode" )
153             let order = Convert.ToInt32( info.Element( "ZOrder" ).Value )
154             where order == index + 1
155             select info.Element( "FileName" ).Value;
156
157          string title = bookTitle.Single(); // gets book title
158
159          // only want title of book, not the rest of the file name
160          title = Path.GetFileName( title );
161
162          // make slight changes to the file name
163          title = title.Replace( ".jpg", string.Empty );
164          title = title.Replace( "pp", "++" );
165          title = title.Replace( "sharp", "#" );
166
167          // display the title on the page
168          return title.ToUpper();
169       } // end method GetTitle
170    } // end class MainPage
171 } // end namespace DeepZoomCoverCollage
```

Fig. 29.22 | **DeepZoomCoverCollage** employs Silverlight's deep zoom (code-behind). (Part 4 of 4.)

Zooming a ***MultiScaleImage***

To zoom in or out with a `MultiScaleImage`, we call its **`ZoomAboutLogicalPoint`** method (lines 74 and 78–79), which takes a zoom factor, an *x*-coordinate and a *y*-coordinate as

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

parameters. A zoom factor of 1 keeps the image at its current size. Values less than 1 zoom out and values greater than 1 zoom in. The method zooms toward or away from the coordinates passed to the method. The coordinates need to be absolute points divided by the entire collage's `Width`. To convert the absolute coordinates raised by a mouse event to these coordinates, we use `MultiScaleImage`'s **`ElementToLogicalPoint`** method (line 70), which takes the `Point`'s absolute coordinates as parameters.

Panning a `MultiScaleImage`

The viewport of a `MultiScaleImage` represents the portion of the image that is rendered on screen. To pan, change the **`ViewportOrigin`** property of the `MultiScaleImage` (line 116). By keeping track of the offset between where the user initially clicked (line 67) and where the user has dragged the mouse, we can calculate where we need to move the origin (lines 110–115) to shift the image. Figures 29.23–29.24 demonstrate what values are returned by various `MultiScaleImage` properties. Assume the "container" of Fig. 29.23 is the viewport while the "image" is the entire collage.

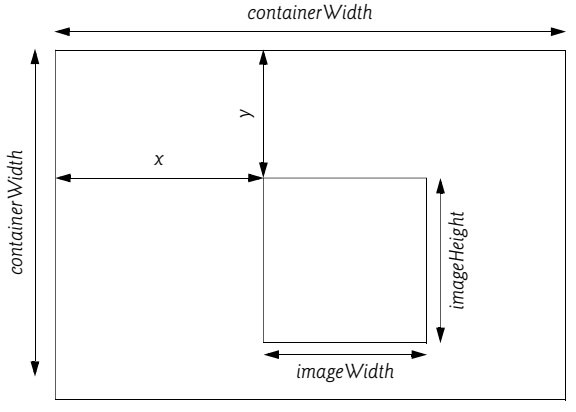


Fig. 29.23 | Various values used to by `MultiScaleImage`'s properties.

Property	Value
`ViewportOrigin`	$\left(\frac{-x}{imageWidth}, \frac{-y}{imageWidth}\right)$
`ViewportWidth`	$\frac{containerWidth}{imageWidth}$
`AspectRatio`	$\frac{imageWidth}{imageHeight}$
`ActualWidth`	$containerWidth$

Fig. 29.24 | Values returned by `MultiScaleImage`'s properties.

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

To determine the new *x*-coordinate of the `ViewportOrigin`, we first find the difference between the *x*-coordinates of the current mouse position (`e.GetPosition(Image).X`) and the mouse position where the user initially clicked (`dragOffset.X`), which we'll refer to as the mouse offset. To convert this value to one we can use for the `ViewportOrigin`, we need to divide it by the width of the collage. The `MultiScaleImage`'s **`ViewportWidth`** returns the ratio of the viewport's width and the collage's width. A `MultiScaleImage`'s **`ActualWidth`** property returns the width of the piece of the collage rendered on-screen (viewport's width). Multiplying the mouse offset by the `ViewportWidth` and dividing by the `ActualWidth` returns the ratio of the mouse offset and the collage's width. We then subtract this value from the `ViewportOrigin`'s original *x*-coordinate to obtain the new value. A similar calculation is performed for the *y*-coordinate (keep in mind we still use `ActualWidth` in this calculation since `ViewportOrigin`'s coordinates are given in terms of the width).

Determining the Title of the Clicked Cover

To determine a clicked image's book title requires the `SparseImageSceneGraph.xml` file created by Deep Zoom Composer. In the **Solution Explorer**, find this XML file in the collection folder we imported and drag the file to your Silverlight deep zoom project so that you can use it in a LINQ query later in the application. The file contains information on where each subimage is located in the collage.

To determine which cover the user clicked, we create a **`Rect`** object (lines 131–133) for each subimage that represents the on-screen area that the image occupies. A `Rect` defines a rectangular area on the page. If the `Point` returned by the mouse-click event is inside the `Rect`, the user clicked the cover in that `Rect`. We can use `Rect` method **`Contains`** to determine whether the click was inside the rectangle. If a cover was clicked, method `SubImageIndex` returns the index of the subimage. Otherwise the method returns `-1`.

A `MultiScaleSubImage`'s properties return the same ratios as a `MultiScaleImage`'s properties (Figs. 29.23–29.24), except that the "container" represents the entire collage while the "image" represents the subimage. Since the `ElementToLogicalPoint` method of a `MultiScaleImage` control returns points based on a scaled coordinate system with the origin at the top-left corner of the collage, we want to create `Rect` objects using the same coordinate system. By dividing the subimage's `ViewportOrigin` by the subimage's `ViewportWidth`, we obtain coordinates for the top-left corner of the `Rect`. To find the `Rect`'s `Width`, we take the inverse of the subimage's `ViewportWidth`. We can then use the subimage's `AspectRatio` to obtain the `Height` from the `Width`.

Next, we use the subimage's index in a LINQ to XML query (in method `GetTitle`) to locate the subimage's information in the `SparseImageSceneGraph.xml` document (lines 151–155). Each subimage in the collage has a unique numeric `ZOrder` property, which corresponds to the order in which the images are rendered on screen—the cover with a `ZOrder` of 1 is drawn first (behind the rest of the covers), while the cover with a `ZOrder` of 12 is drawn last (on top of all other covers). This ordering also corresponds to the order of the subimages in the collection `Image.SubImages` and therefore corresponds with the index that we found in the `SubImageIndex` method. To determine which cover was clicked, we can compare the returned index with the `ZOrder` of each subimage in the collection using our LINQ to XML query. We add 1 to the returned index (line 154), because the indices in a collection start at 0 while the `ZOrder` properties of the subimages start at 1. We then obtain and return the title from the subimage's original file name (lines 157–

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

168) and display the title above the deep zoom image (lines 87–88). If none of the covers were clicked, then no title is displayed (line 92).

29.7 Audio and Video

Silverlight uses the `MediaElement` control to embed audio or video files into your application. A `MediaElement`'s source can be a file stored with the Silverlight application or a source on the Internet. `MediaElement` supports playback in many formats. For a list, see:

```
msdn.microsoft.com/en-us/library/cc189080(VS.95).aspx
```

Silverlight supports high-definition video. Microsoft's Expression Encoder can be used to convert files into a supported format. Other encoders that can convert to Windows media format will work as well, including the free online media encoder at

```
media-convert.com/
```

`MediaElements` can be in one of the following states—`Buffering`, `Closed`, `Paused`, `Opening`, `Playing` or `Stopped`. A `MediaElement`'s state is determined by its **`CurrentState`** property. When in the `Buffering` state, the `MediaElement` is loading the media in preparation for playback. When in the `Closed` state, the `MediaElement` contains no media and displays a transparent frame.

Our **VideoSelector** application (Fig. 29.25) shows some of Silverlight's media-playing capabilities. This application obtains its video sources from a user-created XML file and displays small previews of those videos on the left side of the screen. When you click a preview, the application loads that video in the application's main area. The application plays the audio only for the video in the main area.

```
1  <!-- Fig. 29.25: MainPage.xaml -->
2  <!-- VideoSelector lets users watch several videos at once (XAML). -->
3  <UserControl x:Class="VideoSelector.MainPage"
4     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
5     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
6     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
7     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
8     mc:Ignorable="d">
9
10    <Grid x:Name="LayoutRoot" Background="White">
11       <Grid.ColumnDefinitions> <!-- Defines the page's two columns -->
12          <ColumnDefinition Width="Auto" />
13          <ColumnDefinition />
14       </Grid.ColumnDefinitions>
15
16       <Grid.Resources> <!-- Contains the page's animations -->
17
18          <!-- Fades the main screen in, displaying the new video -->
19          <Storyboard x:Name="fadeIn" Storyboard.TargetName="screen">
20             <DoubleAnimation Storyboard.TargetProperty="Opacity"
21                From="0" To="1" Duration="0:0:0.5" />
22          </Storyboard>
```

Fig. 29.25 | **VideoSelector** lets users watch several videos at once (XAML). (Part 1 of 2.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
23
24          <!-- Fades the main screen out when a new video is selected -->
25          <Storyboard x:Name="fadeOut" Storyboard.TargetName="screen"
26             Completed="fadeOut_Completed">
27             <DoubleAnimation Storyboard.TargetProperty="Opacity"
28                From="1" To="0" Duration="0:0:0.5" />
29          </Storyboard>
30       </Grid.Resources>
31
32       <!-- ListBox containing all available videos -->
33       <ListBox x:Name="previewListBox"
34          SelectionChanged="previewListBox_SelectionChanged">
35          <ListBox.ItemsPanel>
36             <ItemsPanelTemplate>
37                <StackPanel Orientation="Vertical" />
38             </ItemsPanelTemplate>
39          </ListBox.ItemsPanel>
40       </ListBox>
41
42       <!-- Rectangle object with a video brush showing the main video -->
43       <Rectangle x:Name="screen" Grid.Column="1">
44          <Rectangle.Fill>
45             <VideoBrush x:Name="brush" Stretch="Uniform" />
46          </Rectangle.Fill>
47       </Rectangle>
48    </Grid>
49 </UserControl>
```

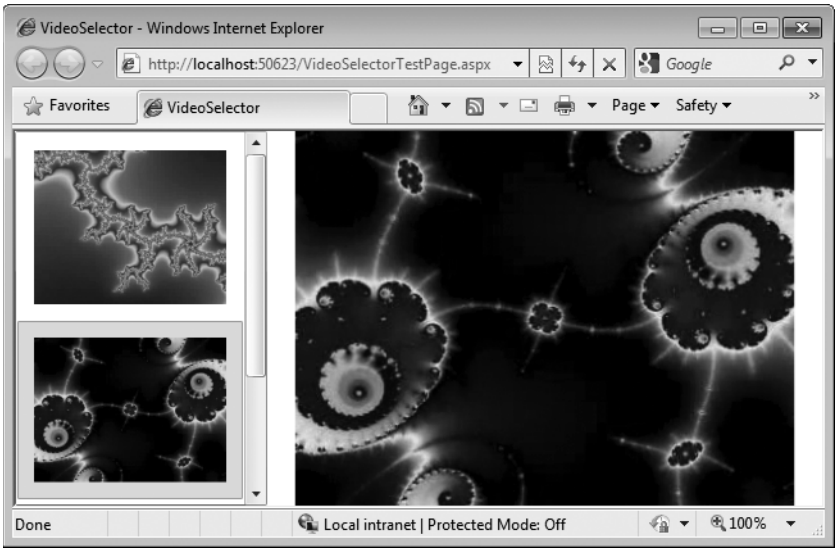


Fig. 29.25 | **VideoSelector** lets users watch several videos at once (XAML). (Part 2 of 2.)

The videos for this example were downloaded from the Wikimedia Commons website (`commons.wikimedia.org`) and are in the public domain. This site contains many

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

images and sound and video files that you can use in your programs—not all items are in the public domain. The videos in the screen capture in Fig. 29.25 were obtained under the science videos section at

```
commons.wikimedia.org/wiki/Category:Science_videos
```

The files were `.ogg` files that we converted to `.wmv` files using the online video converter at `media-convert.com/`.

The application displays one preview video on the side of the page for each source defined in a user-created XML file (discussed shortly). The GUI contains a `Grid` with two `Columns`. The first `Column` contains a `ListBox` that allows you to scroll through previews of the videos (lines 33–40). The second `Column` contains a `Rectangle` element with a `VideoBrush` for its `Fill` (lines 43–47). A **`VideoBrush`** displays a video as a graphics object's `Fill`—similar to an `ImageBrush`. The `SetSource` method of `VideoBrush` takes a `MediaElement` as a parameter and sets the video to be played in the brush.

The `Grid` element contains two `Storyboard Resources`, which contain the main video's fade-in and fade-out animations (lines 19–29). These animations are `DoubleAnimations` that target the `Opacity` property of the `Rectangle` that displays the video. To make the `Rectangle` display the selected video, we'll change the `VideoBrush`'s source to the video the user clicks.

When the page loads, the application performs several initialization tasks. It first loads a new `MediaElement` for each source that is included in the `sources.xml` file (Fig. 29.26). We query this XML file using LINQ to XML. To specify your own list of videos, you must edit our `sources.xml` file, or create a new one and add it to the project. To do this, open a new XML file by right clicking the application project—in this case **VideoSelector**—in the **Solution Explorer** and go to **Add > New Item...**. Select **Visual C#** in the **Categories** section of the window, then select **XML File** in the **Templates** section. Change the file's **Name** to **sources.xml** and click **Add**. Open the file to begin editing it. The sample file in Fig. 29.26 shows the format required to list the sources of the desired videos.

```
1  <?xml version="1.0" encoding="utf-8" ?>
2
3  <!-- Fig. 29.26: sources.xml -->
4  <!-- VideoSelector's list of video sources. -->
5  <videos>
6     <video> <!-- each video child contains a uri source property -->
7        <uri>/newfractal.wmv</uri> <!-- source for first video -->
8     </video>
9     <video>
10       <uri>/fractal.wmv</uri> <!-- source for second video -->
11    </video>
12    <video>
13       <uri>/bailey.wmv</uri> <!-- source for third video -->
14    </video>
15 </videos>
```

Fig. 29.26 | **VideoSelector**'s list of video sources.

The XML document defines a `videos` element that may contain any number of `video` elements. Each `video` element contains a `url` element whose value is the source URL for

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

the corresponding `MediaElement`. Simply replace the value in the `url` tag(s) with the path to your video(s). These videos also need to be included in your **Web Project**'s **ClientBin** if you want to play them from the same location as the Silverlight application. If your source URLs link to online videos, then you'll need to change the `UrlKind` in line 31 (Fig. 29.27). To add the local files, right click the **ClientBin** folder in the **Web Project** associated with your Silverlight application (**VideoSelector.Web**) in the **Solution Explorer** and select **Add > Existing Item...**. Locate the videos you want to add and click **Add**. Now that we've added the necessary files to the project, we can continue with the code-behind file shown in Fig. 29.27.

The **VideoSelector** uses LINQ to XML to determine which videos to display in the side bar. Line 22 defines the `XDocument` that loads `sources.xml`. Lines 25–35 contain a LINQ query that gets each `video` element from the XML file. For each `video` element that has a non-empty `url` element, the query creates a new `MediaElement` with that `url` as its relative `Source`. If your video is in the same location as the application or any subdirectory of that location, you may use a relative `Source` value. Otherwise, you need to use an absolute `Source`, which specifies the full path of the video. We set each element's `Width`, `Margin` and **`IsMuted`** properties to specify how the videos appear and perform when the application loads. Setting a `MediaElement`'s `IsMuted` property to `true` (line 34) mutes its audio—the default value is `False`—so that we do not hear the audio from all videos at once. We then assign the videos to the `ItemsSource` (line 38) of the `ListBox` to display the preview videos.

```
1  // Fig. 29.27: VideoSelector.xaml.cs
2  // VideoSelector lets users watch several videos (code-behind).
3  using System;
4  using System.Linq;
5  using System.Windows;
6  using System.Windows.Controls;
7  using System.Windows.Media;
8  using System.Xml.Linq;
9
10 namespace VideoSelector
11 {
12    public partial class VideoSelectorPage : UserControl
13    {
14       private MediaElement currentVideo = new MediaElement();
15
16       // constructor
17       public VideoSelectorPage()
18       {
19          InitializeComponent();
20
21          // sources.xml contains the sources for all the videos
22          XDocument sources = XDocument.Load( "sources.xml" );
23
24          // LINQ to XML to create new MediaElements
25          var videos =
26             from video in sources.Descendants( "video" )
27             where video.Element( "uri" ).Value != string.Empty
```

Fig. 29.27 | **VideoSelector** lets users watch several videos at once. (Part 1 of 2.)

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

```
28              select new MediaElement()
29              {
30                 Source = new Uri( video.Element( "uri" ).Value,
31                    UriKind.Relative ),
32                 Width = 150,
33                 Margin = new Thickness( 10 ),
34                 IsMuted = true
35              };
36
37           // send all videos to the ListBox
38           previewListBox.ItemsSource = videos;
39        } // end constructor
40
41        // when the user makes a new selection
42        private void previewListBox_SelectionChanged( object sender,
43           SelectionChangedEventArgs e )
44        {
45           fadeOut.Begin(); // begin fade out animation
46        } // end method previewListBox_SelectionChanged
47
48        // change the video if there is a new selection
49        private void fadeOut_Completed( object sender, EventArgs e )
50        {
51           // if there is a selection
52           if ( previewListBox.SelectedItem != null )
53           {
54              // grab the new video to be played
55              MediaElement newVideo =
56                 ( MediaElement ) previewListBox.SelectedItem;
57
58              // if new video has finished playing, restart it
59              if ( newVideo.CurrentState == MediaElementState.Paused )
60              {
61                 newVideo.Stop();
62                 newVideo.Play();
63              } // end if
64
65              currentVideo.IsMuted = true; // mute the old video
66              newVideo.IsMuted = false; // play audio for main video
67
68              currentVideo = newVideo; // set the currently playing video
69              brush.SetSource( newVideo ); // set source of video brush
70           } // end if
71
72           fadeIn.Begin(); // begin fade in animation
73        } // end method fadeOut_Completed
74     } // end class VideoSelectorPage
75  } // end namespace VideoSelector
```

Fig. 29.27 | **VideoSelector** lets users watch several videos at once. (Part 2 of 2.)

The application uses `previewListBox`'s `SelectionChanged` event handler to determine which video the user wants to view in the main area. When this event occurs, we begin the fade-out animation (line 45). After the fade-out animation completes, the appli-

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

cation determines which video was clicked by grabbing `previewListBox`'s `SelectedItem` object and stores it in a `MediaElement` variable (lines 55–56).

When a video has finished playing, it is placed in the `Paused` state. Lines 59–63 ensure that the selected video is restarted if it is in this state. We then mute the audio of the old video and enable the audio of the selected video (lines 65 and 66 respectively). Next, we set the source for the `VideoBrush` of the `Rectangle`'s `Fill` to the selected video (line 69). Finally, we begin the fade-in animation to show the new video in the main area (line 72).

29.8 Wrap-Up

In this chapter, you learned how to use Silverlight (a cross-platform, cross-browser subset of .NET) to build Rich Internet Applications (RIAs) in Visual Web Developer 2010 Express. We began by introducing the **WeatherViewer** application to portray some of the key features of a new Silverlight application. Silverlight and WPF have similar programming environments with slight minor variations. The GUI of any Silverlight page is created by a XAML file in the project. All event handlers and other methods are created in the code-behind files.

With the **WeatherViewer** example, we showed that you can use web services, LINQ to XML and data binding to create a web application with desktoplike capabilities. We also showed you how to create a custom control by using a `UserControl` as a template. Unlike `Styles` and `ControlTemplates`, custom controls allow you to manipulate the control's functionality rather than just the visual aspects. The GUI and code-behind of a custom control are created in their own `.xaml` and `.xaml.cs` files.

We showed you our **FlickrViewer** example, which, similar to the **WeatherViewer**, shows how to use web services to enhance the capabilities of your application—specifically in this example with the `Image` control. This application combines a web service—provided by Flickr—and animations to create a photo-searching website. We also introduced Silverlight's out-of-browser experience capabilities.

You learned about Silverlight's deep zoom capabilities. You saw how to use Deep Zoom Composer and Silverlight to create your own deep zoom application. We showed how to implement zooming, panning, and subimage recognition in the code-behind file of your application using `MultiScaleImage` and `MultiScaleSubImage`.

Silverlight supports audio and video playback using the `MediaElement` control. This control supports embedding Windows media format files into the application. We introduced our **VideoSelector** application to show how to program `MediaElements` in your application. The example also showed the `VideoBrush` control being applied to the `Fill` of a `Rectangle` (applicable to any graphics object) to display the video within the graphic. In the next chapter, we begin presenting our object-oriented design case study.

Summary

Section 29.1 Introduction

- Silverlight™ is Microsoft's platform for Rich Internet Applications (RIAs) and is a subset of the .NET platform.
- RIAs are web applications that offer responsiveness and rich GUI features comparable to those of desktop applications.

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

- Silverlight competes with Adobe Flash and Flex and Sun's JavaFX, and complements Microsoft's ASP.NET and ASP.NET AJAX.
- Silverlight runs as a web-browser plug-in.

Section 29.2 Platform Overview
- Silverlight applications consist of XAML files describing the GUI and code-behind files defining the program logic.
- The subset of .NET available in Silverlight includes APIs for collections, input/output, generics, multithreading, globalization, XML, LINQ and more. It also includes APIs for interacting with JavaScript and the elements in a web page.
- Silverlight applications can be created in .NET languages such as Visual C#, Visual Basic, IronRuby and IronPython.

Section 29.4 Building a Silverlight WeatherViewer *Application*
- Silverlight shares many capabilities with WPF.
- A default Silverlight project contains `MainPage.xaml`, `MainPage.xaml.cs`, `App.xaml` and `App.xaml.cs`. The latter two contain code that is shared across all of the application's pages.
- `UserControl` is Silverlight's primary control, similar to the `Window` control in WPF.
- `UserControl` has a class name specified with the `x:Class` attribute.
- A compiled Silverlight application is packaged as a `.xap` file containing the application and its supporting resources.

Section 29.4.1 GUI Layout
- The main layout controls in Silverlight are `Grid`s, `StackPanel`s and `Canvas`es.
- The `ItemsPanel` of a `ListBox` defines how the collection of items is displayed.
- The `ItemTemplate` of a `ListBox` contains a `DataTemplate` which defines the layout of a particular item in the collection.

Section 29.4.2 Obtaining and Displaying Weather Forecast Data
- The `WebClient` class can be used to invoke a web service.
- The `DownloadStringAsync` method of `WebClient` invokes the web service and takes a `Uri` object containing the web-service address as a parameter.
- The `DownloadStringCompleted` event of `WebClient` is raised when the result of the web-service call has finished downloading.
- The `DownloadStringCompletedEventArgs` parameter of the `DownloadStringCompleted` event handler contains the returned information and any errors that occurred.
- `XDocument`'s `Parse` method converts a `String` to an `XDocument`.
- You can bind data to a `ListBox`'s `ItemsSource` to change the set of items the `ListBox` displays.
- A control has a `Visibility` property that determines whether the control is visible on the page.

Section 29.4.3 Custom Controls
- `Style`s and `ControlTemplate`s can be used to customize the visual aspects of a control.
- Custom controls can be created by using a `UserControl` as a template.
- Custom controls can be programmed similarly to a separate Silverlight application.

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

Section 29.5 Animations and the FlickrViewer

- Some Silverlight animations types include `DoubleAnimations`, `PointAnimations`, and `ColorAnimations`.
- Properties with relative values such as `"*"` and `"Auto"` do not animate.
- A `Storyboard` contains one or more animations that can be invoked by the `Begin` method.
- A `Storyboard` is created as a `Resource` element of a layout control.
- A tag is any user-generated word or phrase that helps organize web content.
- Silverlight's out-of-browser experience enables the user to install a Silverlight web application locally like a desktop application.

Section 29.6 Images and Deep Zoom

- Silverlight's deep zoom allows you to view high-resolution images in a browser at different zoom levels while maintaining quality.
- Deep zoom works by sending only the necessary image information to the client machine.
- Silverlight's `MultiScaleImage` element is used to display deep zoom images. It takes an XML file as its source.
- A `MultiScaleSubImage` contains information on a single image in the deep zoom collage.

Section 29.6.1 Getting Started With Deep Zoom Composer

- Deep Zoom Composer splits a collage into separate images for different zoom levels.
- The **Import** tab allows you to add images on your computer to the collection.
- The **Compose** tab allows you to create the collage using the imported images. There are snapping and alignment tools to help you lay out the images.
- The **Export** tab allows you to export the collection to the format required by `MultiScaleImage`.

Section 29.6.2 Creating a Silverlight Deep Zoom Application

- Exported deep zoom collages need to be included in the **ClientBin** folder of the web project that executes your Silverlight application.
- The `ZoomAboutLogicalPoint` method zooms into or out of a `MultiScaleImage`.
- The viewport of the `MultiScaleImage` represents the part of the image that is currently visible. The `ViewportOrigin` property can be changed to pan the image.
- A deep zoom collage's `SparseImageSceneGraph.xml` file contains information on the location of each subimage.
- The `ZOrder` property defines the order in which the subimages are rendered. This value is unique to each subimage.
- The `Rect` class represents a rectangular area on the page. The class's `Contains` method can be used to determine whether a point is in a `Rect`'s area.

Section 29.7 Audio and Video

- Silverlight uses the `MediaElement` control to embed audio or video files into the application.
- A `MediaElement` supports windows media audio and video sources.
- Microsoft's Expression Encoder can convert videos into a supported windows media format.
- A `VideoBrush` can be applied to the `Fill` property of a graphics element. This imposes the video onto the background of that graphics element.
- The `IsMuted` property of a `MediaElement` specifies whether the media's audio is audible.
- `MediaElements` can be in the `Buffering`, `Closed`, `Opening`, `Playing`, `Paused` or `Stopped` states.

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

Terminology

`ActualWidth` property of `MultiScaleImage` control
`Contains` method of class `Rect`
`CurrentState` property of `MediaElement`
custom control
deep zoom
Deep Zoom Composer
`EasingFunction`
`ElementToLogicalPoint` method
`IsMuted` property of `MediaElement`
`Margin` property of `Border` control
multimedia
`MultiScaleImage` control
`MultiScaleSubImage` control
out-of-browser experience
`Parse` method of `XDocument`
`Rect` structure
Rich Internet Application (RIA)
Silverlight
Silverlight Runtime
tag
tagging
`UserControl`
`VideoBrush` control
`ViewportWidth` property of `MultiScaleImage` control
`x:Class` attribute of a `UserControl`
`x:Name` attribute of a control
`.xap` file extension
`zoomAboutLogicalPoint` method

Self-Review Exercises

29.1 Say whether the statement is *true* or *false*. If it is *false*, explain why.

a) Silverlight employs all of the same functionality as WPF but in the form of an Internet application.
b) Silverlight competes with RIA technologies such as Adobe Flash and Flex and Sun's JavaFX, and complements Microsoft's ASP.NET and ASP.NET AJAX.
c) The `.xap` file contains the application and its supporting resources and is packaged by the IDE.
d) Silverlight's template control is `Window`.
e) Users can create custom controls by using the Silverlight `Style` and `ControlTemplate` controls.
f) When you call `WebClient`'s `DownloadStringAsync` method, the user can still interact with the application while the string is downloading.
g) A deep zoom image is just a high-resolution image.

29.2 Fill in the blanks with the appropriate answer.

a) The three basic animation controls are _____, _____, and _____.
b) An object of class _____ can be used to invoke a web service.
c) The `XDocument` method _____ converts a `String` containing XML into an object that can be used with LINQ to XML.
d) Namespace _____ is required to use LINQ to XML in your application.
e) When a `MediaElement` has finished playing, it is in the _____ state.
f) The three most used layout controls for Silverlight are _____, _____ and _____.
g) The _____ of a `MultiScaleImage` represents the area of the deep zoom image that the user is currently viewing.

Answers to Self-Review Exercises

29.1 a) False. Silverlight is a subset of WPF, therefore it does not contain all of the same functionality as a WPF application. b) True c) True d) False. Unlike WPF applications, the template control for Silverlight applications is the `UserControl`. e) False. While `Styles` and `ControlTemplates` can be used to customize existing controls, `UserControl` is the template used to create cus-

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

tom controls. f) True g) False. A deep zoom image is really a collection of images. Deep Zoom Composer separates your original collage into these images, which are sent over the Internet to the client machine.

29.2 a) `DoubleAnimation`, `PointAnimation`, `ColorAnimation`. b) `WebClient`. c) `Parse`. d) `System.Xml.Linq`. e) `Paused`. f) `Grid`, `StackPanel`, `Canvas`. g) viewport.

Exercises

29.3 *(Enhanced* ***WeatherViewer*** *Application)* Modify the **WeatherViewer** application (Section 29.4) to display the name of the city and state for the zip code input by the user. These are specified by elements named `City` and `State` in the XML returned by the web service. To update the application's layout, you'll need to:

a) Add a new `RowDefinition` in the main `Grid` to accommodate the additional information.
b) Change the `ListBox`'s `Grid.Row` attribute so that it appears in the bottom row of the main `Grid`. Also, increase the `RowSpan` of the `WeatherDetailsView`.
c) Insert a horizontal `StackPanel` before the `ListBox`. This `StackPanel` should contain four `TextBlocks`—two to label the city and state, and two to display the values for the city and state.
d) Add the `CityState` class (provided in the exercises folder with this chapter's examples) to your project. This class consists of `String` properties `City` and `State`.
e) Use LINQ to XML to create a `CityState` object containing the values of the elements named `City` and `State` in XML returned by the web service.
f) Set the `DataContext` of the `StackPanel` in *Step c* to the `CityState` object from *Step e*.
g) Use binding markup extensions to bind the city and state information to the appropriate `TextBlocks` you created in *Step c*.

29.4 *(Length/Distance Converter)* The website `www.webservicex.net` provides several useful web services, including a length/distance converter. Use this web service and the techniques you learned in this chapter to create a Silverlight length/distance converter. You can invoke the web service with a URL of the form:

```
http://www.webservicex.net/length.asmx/ChangeLengthUnit?LengthValue=100&
   fromLengthUnit=Inches&toLengthUnit=Centimeters
```

If you try the preceding URL in your web browser, you'll see that 100 inches converts to 254 centimeters. The XML returned by the web service appears as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<double xmlns="http://www.webserviceX.NET/">254</double>
```

The element `double` contains the result. The web-service method `ChangeLengthUnit` requires three parameters named `LengthValue`, `fromLengthUnit` and `toLengthUnit`, each of type `String`. The complete list of values you can supply for the `fromLengthUnit` and `toLengthUnit` parameters can be found by looking at the bottom of the following web page in the section labeled **WSDL Schema**:

```
http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=71
```

The user should be able to enter a number in a `TextBox`, then select the two units of measurement from `ListBoxes` and click a `Button` to invoke the web service. You can bind each `ListBox`'s `ItemsSource` property to an array of `Strings` containing the complete list of measurement names.

29.5 *(Enhanced* ***VideoSelector*** *Application)* Modify the **VideoSelector** application (Section 29.7) to include a **Play/Pause** and a **Stop** button. When the **Play/Pause** button is clicked, its text should change from **Play** to **Pause** or from **Pause** to **Play**, respectively. Place the buttons below the currently selected video. These buttons should allow the user to play, pause or stop the current video.

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

29.6 Combine capabilities of the **FlickrViewer** application (Section 29.5) and the **DeepZoomCoverCollage** application (Section 29.6.2) to create a Deitel Book-Cover Viewer with the following features. The `ListBox` should display thumbnails of the book-cover images (provided in the exercises folder with this chapter's examples). When the user selects a book-cover thumbnail from the `ListBox`, the application should display a deep zoom image of that cover. Use the code from the **DeepZoomCoverCollage** application to implement the zooming and panning. You'll need to change the code to use keys other than the up-arrow and down-arrow, because those buttons change the selection in the `ListBox`. We suggest the up arrow key for zooming in and the down arrow key for zooming out.

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.

© 2011 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved.