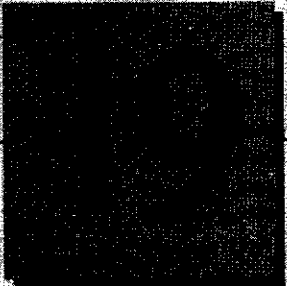


Exercises

18.3 For the `coverViewer` application, add effects to the `ImageRenderer` of the thumbnail images so that the image zooms in when you roll over it (`rollOverEffect`) using the `Zoom` effect, zooms back out when you roll back out (`rollOutEffect`) using the `Zoom` effect, and glows red when you click it (`mouseDownEffect`) using the `Glow` effect. Learn how to configure these effects by referring to Adobe's Flex 2 Language Reference at livedocs.adobe.com/Flex/201/langref/mx/effects/Zoom.html and livedocs.adobe.com/Flex/201/langref/mx/effects/Glow.html. [The effects must be defined locally in the `ImageRenderer`, inside the `VBox`.] An example of how your solution may look is available at test.delta1.com/examples/flex4/Flex/CoverViewerExercise/.

18.4 Combine the XML-handling techniques of the `addressBook` example with the `coverViewer` example to display the 20 most recent photos from Flickr's Public RSS feed. See www.flickr.com/services/feeds/docs/photos_public/ for information on how to use the RSS feed, and Chapter 14 for more information on XML in general. The application should show thumbnails of the images at the bottom, as well as a large version of the selected image. You should set the width and height of the thumbnails to fit in the thumbnails bar. The thumbnails should be labeled with the image's title, and the large image should be accompanied by the image's title and author. Include a button that refreshes the RSS feed and loads the most recent images. You may find it beneficial to use an `ArrayCollection` similar to that in the `weatherChart` example (Fig. 18.23) to store each image's source and title. One feature you may want to include is the ability for the user to search for specific tags by adding `tags=stringOfText` to the end of the RSS URL. When string of tags are added by your own string, the RSS feed will return only images containing those tags. Other features you may want to include are the ability for the user to choose how many photos to display and to click the author's name to open the Flickr homepage. An example of how your solution may look is available at test.delta1.com/examples/flex4/Flex/FlickrPhotoViewerExercise/.

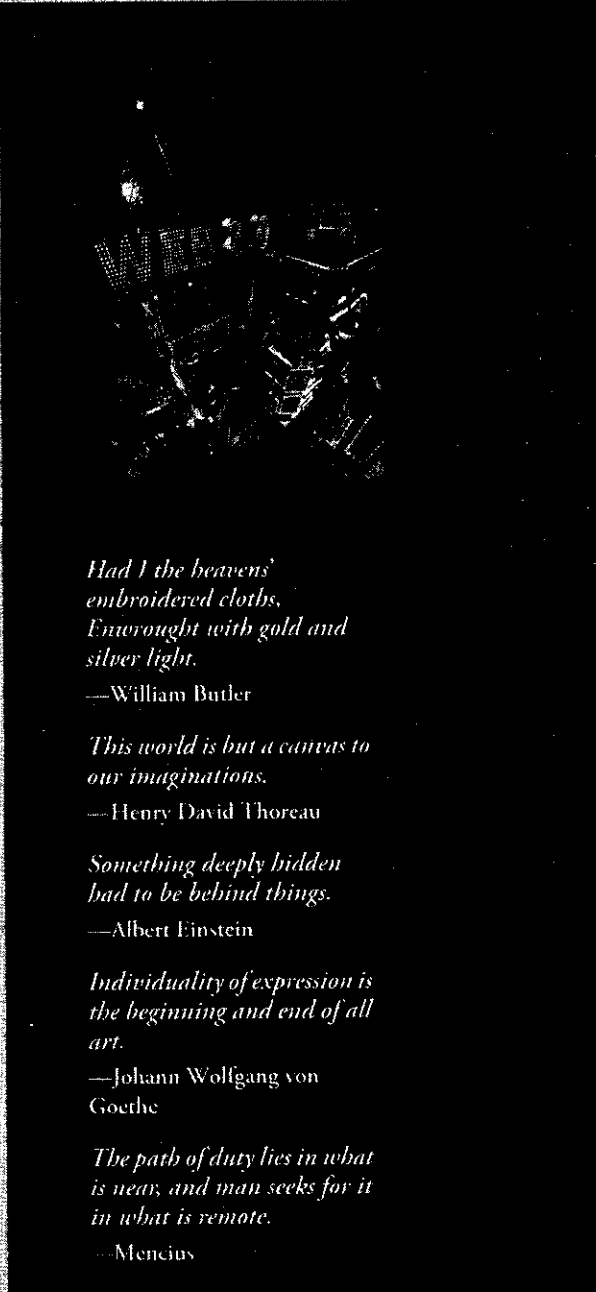


Microsoft Silverlight™ and Rich Internet Applications

OBJECTIVES

In this chapter you will learn:

- What Silverlight is and what its capabilities are.
- The differences between Silverlight 1.0 and 1.1.
- To create user interfaces in XAML.
- To embed multimedia in a Silverlight application.
- To program for Silverlight with JavaScript.
- To embed Silverlight in web pages.
- To host Silverlight applications online with Microsoft's Silverlight Streaming Service.
- To program for Silverlight with .NET languages, specifically Visual Basic.
- To parse RSS feeds in Silverlight 1.1.



*Had I the heavens'
embroidered cloths,
Emorought with gold and
silver light.*

—William Butler

*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

*The path of duty lies in what
is near, and man seeks for it
in what is remote.*

—Mencius

- 19.1 Introduction
- 19.2 Platform Overview
- 19.3 Silverlight 1.0 Installation and Overview
- 19.4 Creating a Movie Viewer for Silverlight 1.0
 - 19.4.1 Creating a User Interface In XAML Using Expression Blend
 - 19.4.2 Using Storyboards
 - 19.4.3 Creating Controls
 - 19.4.4 Using JavaScript for Event Handling and DOM Manipulation
- 19.5 Embedding Silverlight in HTML
- 19.6 Silverlight Streaming
- 19.7 Silverlight 1.1 Installation and Overview
- 19.8 Creating a Cover Viewer for Silverlight 1.1 Alpha
- 19.9 Building an Application with Third-Party Controls
- 19.10 Consuming a Web Service
 - 19.10.1 Consuming the HugeInteger Web Service
- 19.11 Silverlight Demos, Games and Web Resources

Summary | Terminology | Self-Review Exercises | Exercises

19.1 Introduction

Silverlight™, formerly code named “Windows Presentation Foundation Everywhere (WPF/E),” is Microsoft’s platform for Rich Internet Applications (RIAs). It is designed to complement Ajax and other RIA technologies, such as Adobe Flash and Flex, Sun’s JavaFX and Microsoft’s own ASP.NET Ajax. Silverlight currently runs as a browser plug-in for Internet Explorer, Firefox and Safari on recent versions of Microsoft Windows and Mac OS X. In addition, developers from the Mono project (www.mono-project.com) are developing an open-source implementation of Silverlight for Linux distributions called Moonlight.

Microsoft announced Silverlight 1.0 Beta and 1.1 Alpha at the 2007 MIX conference (www.visitmix.com), Microsoft’s annual conference for web developers and designers. The demos were compelling, and many technology bloggers who attended the conference blogged about Silverlight’s excitement and potential. Since then, Microsoft has continued developing and enhancing Silverlight. At the time of this writing, Silverlight is currently available in version 1.0 Release Candidate and version 1.1 Alpha Refresh.

Despite the generally unstable nature of alpha-level software, we felt compelled to include examples using the Silverlight 1.1 Alpha Refresh because of Silverlight 1.1’s potential to become an important RIA development platform. Silverlight 1.1 is still early in its development cycle, so you may encounter bugs while running this software. Also, it is possible that Silverlight 1.1 will change substantially in future releases, breaking our 1.1 based example applications. For updated examples, please visit our website for this book at www.deitel.com/books/iw3http4/. For information on the latest version(s) of Silverlight and to find additional Silverlight web resources, please visit our Silverlight Resource Center at www.deitel.com/silverlight.

19.2 Platform Overview

Silverlight applications consist of a user interface described in **Extensible Application Markup Language (XAML)** and a **code-behind file** (or files) containing the program logic. XAML (pronounced “zammel”) is Microsoft’s XML vocabulary for describing user interfaces and is also used in Microsoft’s Windows Presentation Foundation (WPF)—the preferred user-interface technology of the .NET Platform as of version 3.0.

Silverlight currently runs in Internet Explorer 6+ and Firefox 1.5.0.8+ on Windows XP SP2 and Vista, as well as Safari 2.0.4+ and Firefox 1.5.0.8+ on Mac OS X. Support for Windows 2000 and for the Opera browser is planned in a future release.

Silverlight 1.0 focuses primarily on media and supports programming only in JavaScript. Its primary purpose is to take advantage of the increasing popularity of web-based video to drive user adoption—it is well known that users are willing to install software to watch video. Microsoft also provides a service called Silverlight Streaming that allows you to distribute video-based Silverlight applications for free.

When Silverlight 1.1 is released, computers with Silverlight 1.0 will automatically be upgraded. This could immediately make Silverlight 1.1 a widespread platform for RIA development. Silverlight 1.1’s key benefit is that it adds an implementation of the .NET runtime, allowing developers to create Silverlight applications in .NET languages such as Visual Basic, Visual C#, IronRuby and IronPython. This makes it easy for developers familiar with .NET programming for Windows to create applications that run in a web browser. Two of our 1.1 Alpha Refresh examples borrow their user interfaces and code from examples in our *Visual Basic 2005 How to Program, 3/e* textbook. This straightforward conversion was made possible by Silverlight 1.1’s .NET runtime and a set of third-party Silverlight user-interface controls (available at www.netikatech.com) designed to replicate the standard Windows Forms controls. Microsoft plans to implement their own built-in set of controls in a future release of Silverlight 1.1. Version 1.1 also provides a substantial performance improvement over 1.0 because .NET code is compiled by the developer then executed on the client, unlike JavaScript, which is interpreted and executed on the client at runtime. For a detailed feature comparison of 1.0 Release Candidate and 1.1 Alpha Refresh, visit silverlight.net/GetStarted/overview.aspx.

19.3 Silverlight 1.0 Installation and Overview

You can download the Silverlight 1.0 Release Candidate plug-in from www.microsoft.com/silverlight/install.aspx. After installing the plug-in, go to silverlight.net/themes/silverlight/community/gallerydetail.aspx?cat=1 and try some of the sample applications. We list many other demo websites in Section 19.11.

We developed our Silverlight 1.0 application using Microsoft’s **Expression Blend 2**, a WYSIWYG editor for XAML user interfaces. You can download a free trial of Expression Blend 2 from

www.microsoft.com/Expression/products/download.aspx?key=blend2preview

Follow the instructions on the web page to install the software. Note that Expression Blend runs only on Windows XP SP2 and Vista. Also, note that you do not need to install Visual Studio 2005 Express.

19.4 Creating a Movie Viewer for Silverlight 1.0

Our first example application is a movie viewer (Fig. 19.1) that plays **Windows Media Video (WMV)** videos. This example runs on Silverlight 1.0 Release Candidate, and the user interface was created using Expression Blend 2 August Preview. The XAML was generated primarily by Expression Blend. We discuss the XAML as we show you how to build the user interface.

The movie viewer's GUI includes play/pause, stop and full-screen buttons, a timeline with a marker at the current time, a volume control and thumbnails of other videos that you can view. The timeline also shows the percentage of the video that has been downloaded. In this example, you'll learn to create user interfaces in XAML and to use JavaScript to handle events. We'll also demonstrate how to use JavaScript to manipulate the **Silverlight DOM (Document Object Model)**. You can test a live version of this application at test.deitel.com/examples/iw3http4/silverlight/MovieViewer/index.html.

To create the project in Expression Blend, open Expression Blend and select **New Project** in the **Project** tab. To create a Silverlight 1.0 application, select **Silverlight Application (JavaScript)**. Name the project **MovieViewer** and select the location where you would like to save it.

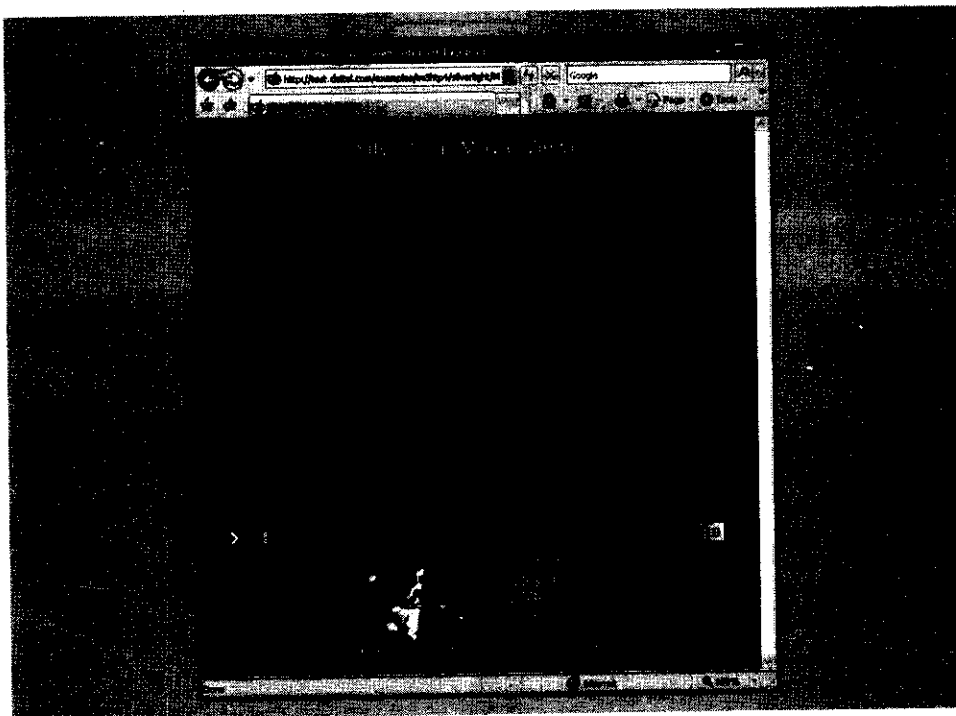


Fig. 19.1 | Silverlight Movie Viewer.

19.4.1 Creating a User Interface In XAML Using Expression Blend

To show how XAML works, we first create elements in Expression Blend, then discuss the corresponding generated XAML in `Scene.xaml` (which you'll see in Fig. 19.12).

Canvas Elements

The root element of the XAML file is a Canvas element. A **Canvas element** acts as a container for other user interface elements and controls their position. The parent Canvas element is created when you create a new Silverlight project in Expression Blend. The parent Canvas has a default Name of Page, Width of 640 px and Height of 480 px. The **Name attribute** provides an ID to access the element programmatically. The Canvas's properties can be edited in the **Properties** panel (Fig. 19.2). Additional Canvas elements can be created in Expression Blend using the Canvas tool in the toolbar, shown in Fig. 19.3. The XAML can be manually edited by selecting **XAML** in Expression Blend's **View** menu.

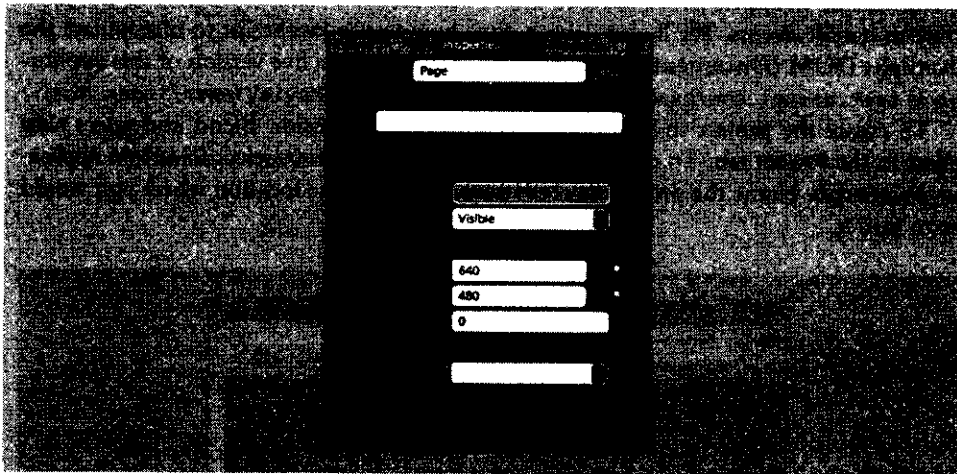


Fig. 19.2 | Expression Blend's **Properties** inspector.

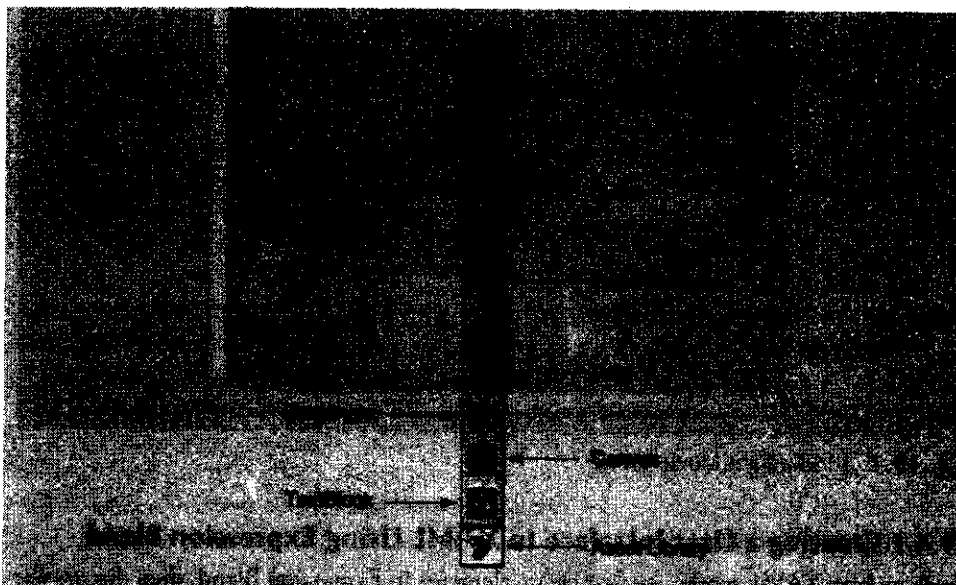


Fig. 19.3 | Expression Blend's toolbar.

19.4.2 Using Storyboards

The **Storyboard** element allows you to define animations. In Expression Blend, you can create a Storyboard by opening the **Open, create or manage Storyboards** panel and clicking the **Create new Storyboard** button (Fig. 19.4). Select the **Create as a Resource** checkbox (Fig. 19.5). This enables you to start the Storyboard from anywhere in the application's JavaScript at any time (as opposed to starting the Storyboard automatically when the application loads). Name the Storyboard `timelineTimer` and click **OK**. This Storyboard will be used as a timer, because a dedicated timer object does not exist in Silverlight 1.0. A Storyboard must have a target object, so we will create an invisible object. Create a **Rectangle** of any size in any location using the **Rectangle** tool, name it `invisibleRectangle`, then set its **Visibility** to **Collapsed** using the **Properties** panel. Move the timeline playhead to 0.1 seconds, then click the **Record Keyframe** button (Fig. 19.6). In this

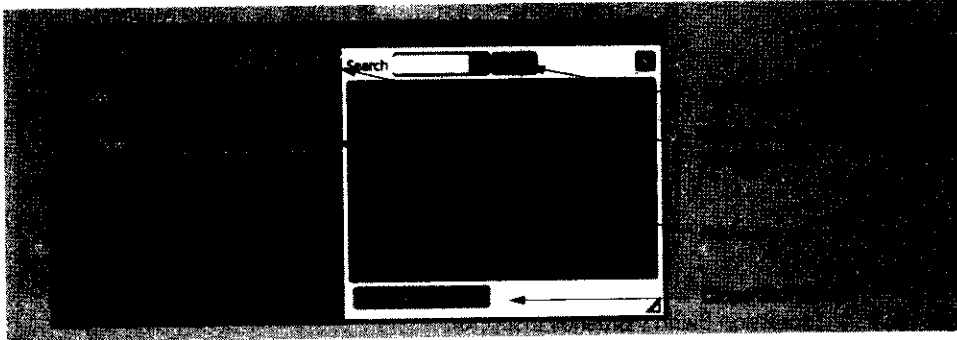


Fig. 19.4 | Expression Blend's **Objects and Timeline** inspector.

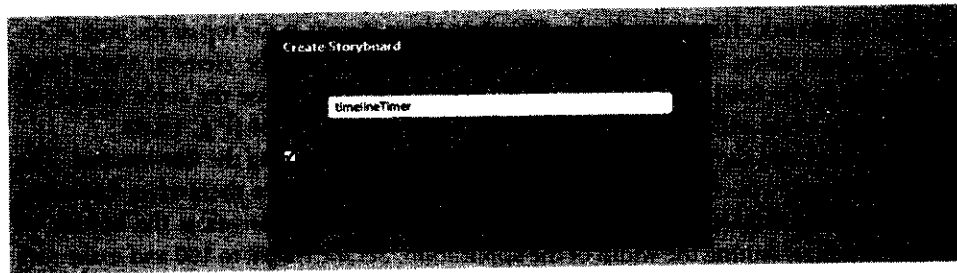


Fig. 19.5 | Expression Blend's **Create Storyboard** dialog box.

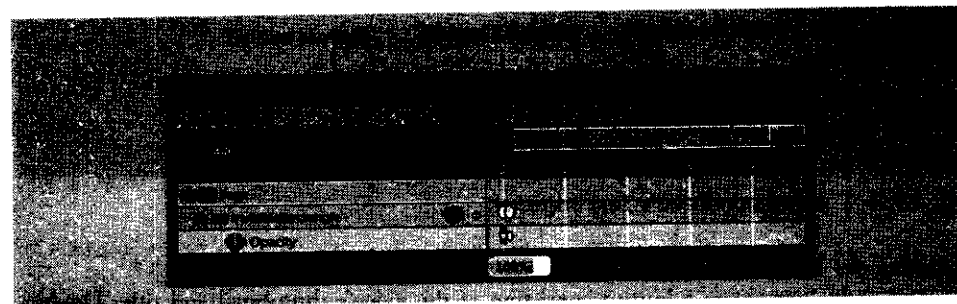


Fig. 19.6 | **Objects and Timeline** inspector showing the `TimelineTimer` Storyboard.

new keyframe, change any property of the rectangle. If you create the keyframe without changing a property, the Storyboard will not do anything. Close the Storyboard by opening the **Open, create or manage Storyboards** menu and clicking **Close Storyboard**.

Expression Blend provides the **Gradient brush tool** (Fig. 19.7) to visually create and modify gradients. First, use the **Selection** tool to select the Page Canvas in the design area. Then, select the **Gradient brush** for the **Background** and select the gradient slider on the right (Fig. 19.7). Change the red, green and blue values to 71.

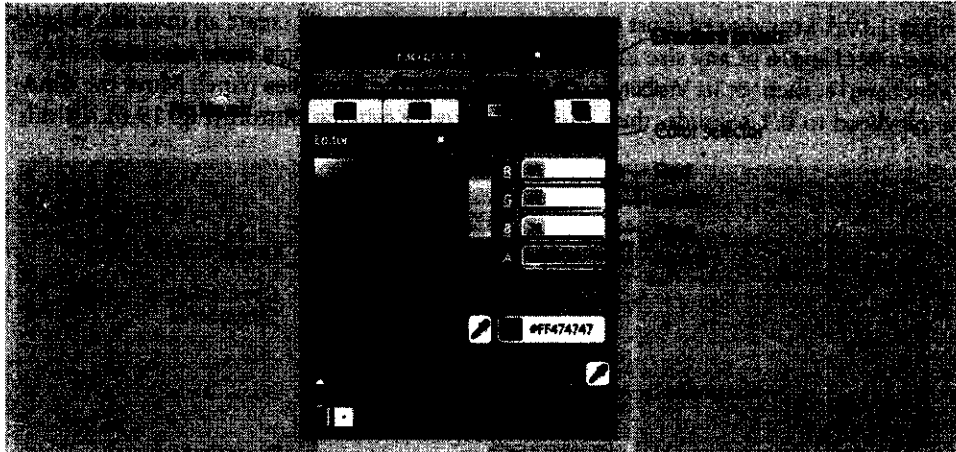


Fig. 19.7 | Expression Blend's **Brushes** inspector.

19.4.3 Creating Controls

We use a **TextBlock** element to display **Silverlight Movie Viewer** at the top of the page. Create this element using Expression Blend's **TextBlock** tool, name the element **titleText**, then change to the **Solid color** brush in the **Brushes** inspector and use the color selector to make the text white. Adjust the text size in the **Text** inspector to **24** (Fig. 19.8).

Next, we create another Canvas element called **controls**, using the **Canvas** tool. The **controls** Canvas will contain the application's buttons, which are themselves Canvases. This Canvas is a child of the Page Canvas element. Create this Canvas at the bottom of the application, spanning the width of the application. Set the **Height** to **160** and make sure that the Canvas is at the bottom of the application by moving it until it snaps into place.

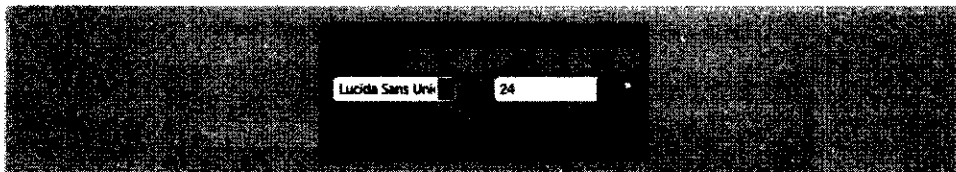


Fig. 19.8 | Expression Blend's **Text** inspector.

Creating the Video Thumbnail Buttons

For each video, we create a button consisting of the video's thumbnail and title. Double-click the **controls** Canvas with the **Selection** tool to activate it, then create four new Canvases

in the controls Canvas. Set each Canvas's **Width** to 120 and **Height** to 114. In each Canvas, create an **Image** element, with the **Source** attribute pointing to the video's thumbnail JPEG image, for example `baileyThumb.jpg` (Fig. 19.9). You can select the **Image** tool by clicking the **Asset Library** button (Fig. 19.3), checking **Show All** and selecting **Image** (Fig. 19.10). Set the Image's **Width** and **Height** to 120 and 90, and place it at the top of the Canvas. Add a **TextBlock** containing the text `Crazy Dog`. Do the same for the other three Canvases, setting the Image's Sources to `featherAndHammerThumb.jpg`, `apollo15LaunchThumb.jpg` and `F35Thumb.jpg`. The **TextBlocks** for the three Canvases should contain `Gravity`, `Apollo 15` and `F35 Landing`, respectively. Name the Canvases `crazyDogButton`, `gravityButton`, `apollo15Button`, and `f35Button`, and space them evenly across the controls Canvas. Finally, to make each of these Canvases appear to act like a button, we will set their **Cursor** properties to **Hand** in the **Common Properties** inspector. This way, the user's cursor will change to a hand when the cursor is moved over each button Canvas.

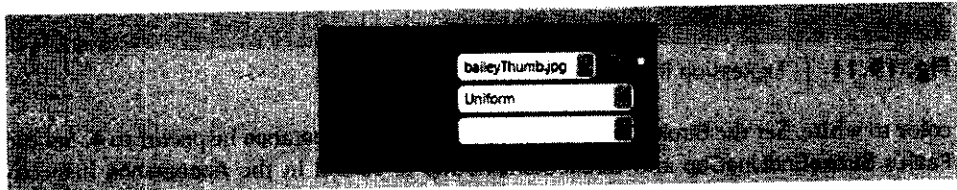


Fig. 19.9 | Expression Blend's **Common Properties** inspector for an **Image** element.

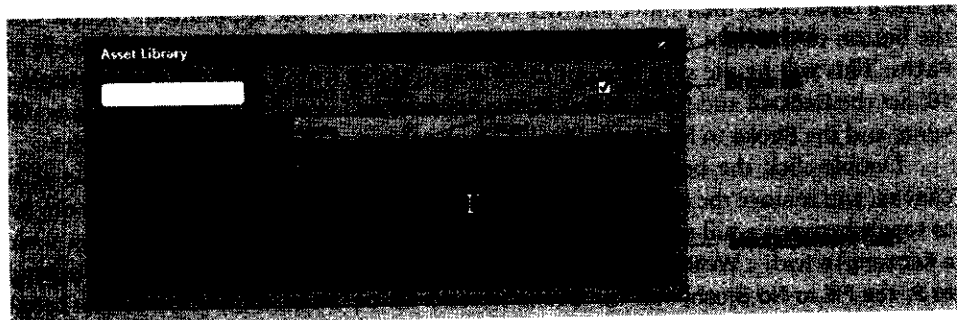


Fig. 19.10 | Expression Blend's **Asset Library**.

Creating the Video Playback Control Buttons

Next, we create a play button, a stop button, a pause button, and a full-screen button. These buttons are all contained in the controls Canvas. To create the play button, first create a Canvas named `playButton`, then set its **Width** and **Height** to 30. Set the **RadiusX** and **RadiusY** to 2 to give the button rounded corners. These properties are located in the **advanced properties** section of the **Appearance** inspector (Fig. 19.11). Inside this Canvas, use the **Rectangle** tool to draw a **Rectangle** with the same width and height as the Canvas, and set its background to a gradient going from dark blue to light blue. Then, using the **Pen** tool, draw two **Paths** to make an arrow pointing right (a play button). Use the **Pen** tool to click once at each endpoint of the line. After drawing each line, use the **Select** tool to move the line into place. The **Path** element allows you to draw shapes that include curves and arcs, but here we are just using it to draw simple lines. Set each **Path**'s **Stroke**

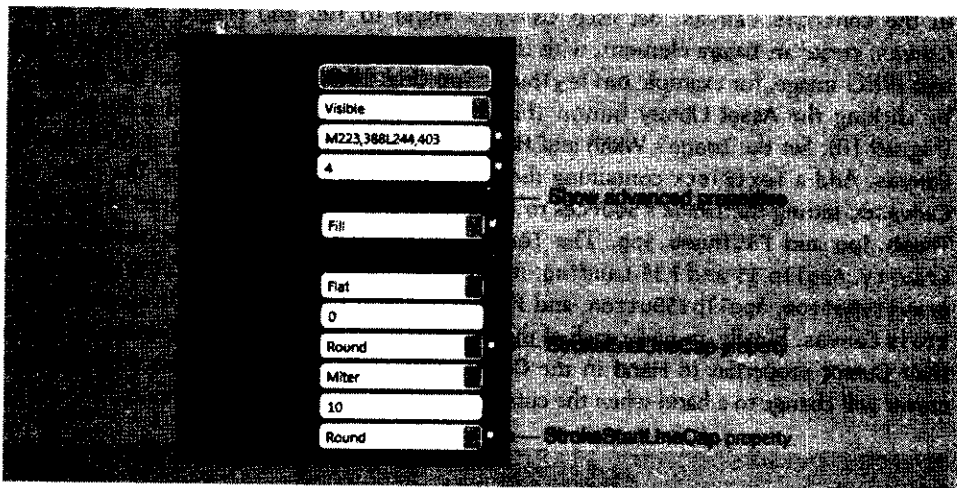


Fig. 19.11 | Expression Blend's Appearance inspector.

color to white. Set the **StrokeThickness** property in the **Appearance** inspector to 4. Set each Path's **StrokeEndLineCap** and **StrokeStartLineCap** to **Round** in the **Appearance** inspector. Finally, set the **Cursor** property of the `playButton` Canvas to **Hand**.

Copy the `playButton` Canvas and paste it three times. Move one copy just to the right of the `playButton`, then move another to the right side of the application. Double-click the button to the right of the `playButton` to make it the active Canvas, and remove the Paths. This will be the stop button. Draw a **Rectangle** with a **Width** of 14 and **Height** of 18. Set the **RadiusX** and **RadiusY** to 2 in the **Appearance** inspector, then set the **Fill** to solid white and the **Stroke** to **No Brush**. Finally, set the **Name** of the Canvas to `stopButton`.

Double-click the button at the far right of the application to make it the active Canvas, and remove the Paths. This will be the full-screen button that will enable the user to toggle between a full-screen view and a browser window view of the application. Draw a **Rectangle** with a **Width** and **Height** of 22. Set the **RadiusX**, **RadiusY** and **StrokeThickness** to 2, the **Fill** to **No Brush**, and the **Stroke** to solid white. Then, draw a second **Rectangle** with a **Width** and **Height** of 10, starting at the bottom-left of the previous **Rectangle**. Give this **Rectangle** the same **StrokeThickness** and **Fill** and **Stroke** colors as the larger **Rectangle**. Finally, name this Canvas `fullscreenButton`.

Double-click the button that is still on top of the `playButton` to make it the active Canvas, and remove the Paths. This will be the pause button. Draw two vertical paths with the same properties as the paths in the `play` button, and space them apart by a few pixels. Name this Canvas `pauseButton` and set its **Visibility** attribute to **Collapsed** (i.e., hidden). We'll programmatically display this button when it is needed.

The application displays the current time of the video in *hh:mm:ss* format in the `timeText` **TextBlock**, located inside the `timeCanvas` Canvas. To create this element, first create a Canvas named `timeCanvas` to the left of the full-screen button, and give it a **Width** of 75 and **Height** of 23. Inside this Canvas, create a **Rectangle** that takes up the entire Canvas. Set this **Rectangle**'s **RadiusX** and **RadiusY** to 2, and its **StrokeThickness** to 1. Set the **Stroke** color to solid black, and the **Fill** to a gradient going from grey to white. Create a **TextBlock** named `timeText` using the **TextBlock** tool, and set its initial text value to "00:00:00". Use

the default font (**Lucida Sans Unicode**) and font size (**14.667**). This `TextBlock`'s text value will be updated programmatically in our JavaScript code-behind file.

Creating the Volume and Timeline Controls

The application allows the user to select the volume level through a volume control. To create this control, first create a `Canvas` named `volumeCanvas` with a **Width** of **15** and a **Height** of **30** to the right of the full-screen button. Use the **Rectangle** tool to create a vertical `Rectangle` (the slider). Give the `Rectangle` a **Width** of **4** and **Height** of **30**. Set the **Fill** of the `Rectangle` to light grey. Set the **Stroke** of the `Rectangle` to black, with a **StrokeThickness** of **1**.

Use the **Rectangle** tool to create a horizontal `Rectangle` (to mark the current volume). Give the `Rectangle` a **Width** of **14** and **Height** of **2**. Set its **Fill** to white, its **Stroke** to **No Brush** and its **Opacity** to **50%** (in the **Appearance** inspector). Center the horizontal `Rectangle` on the center of the vertical `Rectangle`. Finally, name the horizontal `Rectangle` `volumeHead` and the `Canvas` `volumeCanvas`.

Next, we create a video timeline that serves two purposes. The timeline acts as a progress bar while the video is being downloaded. This is accomplished by having a dark grey `Rectangle` with a **Width** of **400** located underneath a light grey `Rectangle` with an initial **Width** of **0**. The light `Rectangle`'s **Width** is set programmatically in the JavaScript code-behind file to indicate the download progress. Also, a vertical `Rectangle` acts as the playhead, indicating the current playback progress of the video. To create the video timeline, first create a `Canvas` named `timelineCanvas` to the right of the stop button. Give this `Canvas` a **Width** of **400** and a **Height** of **20**, and a **Cursor** of **Hand**. Inside this `Canvas`, create a `Rectangle` named `timelineRectangle` with a **Width** of **400** and a **Height** of **4**. Set its **StrokeThickness** to **1**, its **Fill** to dark grey and its **Stroke** to black. Center the `Rectangle` vertically, then copy and paste the `Rectangle`. Name the copy `downloadProgressRectangle`, set its **Fill** to a lighter grey and set its **Width** to **0**. Note that because `downloadProgressRectangle` appears after the `timelineRectangle` in the **Objects and Timeline** inspector, it appears on top of the `timelineRectangle`. You can also specify the *z*-order of elements (discussed in Section 5.6) using an object's **ZIndex** attribute. Higher **ZIndex** integer values position the element closer to the foreground and in front of other elements with smaller **ZIndex** values.

Create a `Rectangle` named `playHead` with a **Width** of **2** and a **Height** of **20**. Place this `Rectangle` at the far left of the `Canvas` and center it vertically. Set this `Rectangle`'s **Fill** to **No brush**, its **StrokeThickness** to **1**, its **Stroke** to white, and its **Opacity** to **50%**.

Using a MediaElement to Display Audio/Video

The `MediaElement` allows you to include video and/or audio in your Silverlight application. It supports WMV/VC-1 (including high-definition video), WMA and MP3 formats.

First, create a `Canvas` named `movieViewCanvas` and set its **Height** to **260** and **Width** to **640**. Inside the `Canvas`, add a `MediaElement` named `movieMediaElement`. To access the `MediaElement` tool, click the **Asset Library** button (Fig. 19.3), check **Show All** and select `MediaElement` (Fig. 19.10). Set the `MediaElement`'s **Width** and **Height** to those of the `Canvas`. Set the **Source** attribute to point to `bailey.wmv` in the **Media** inspector.

This `Canvas` also contains a **Play** button overlaid on the video. First, create a `Canvas` named `playOverlayCanvas` with an **Opacity** of **60%**, a **Width** of **200** and a **Height** of **180**. Inside this `Canvas`, create a `Rectangle` with the same **Width** and **Height** as the `Canvas`, a **Fill** of black, a **Stroke** of **No brush**, and a **RadiusX** and **RadiusY** of **40**. Create an `Ellipse` using

the **Ellipse** tool with a **Width** of 100 and a **Height** of 100. Set its **Fill** to **No brush**, its **Stroke** to white, and its **StrokeThickness** to 6. In the middle of this **Ellipse**, draw two **Paths** in the shape of a right arrow, both with a **Width** and **Height** of 30, a **StrokeThickness** of 6, and a **StrokeStartLineCap** and **StrokeEndLineCap** of **Round**. Underneath the **Ellipse**, create a **TextBlock** containing the text **Play**. Set the font size to 36 in the **Text** inspector.

Finally, set the **playOverlayCanvas** **Visibility** attribute to **Collapsed**, since we will show this **Canvas** programmatically.

Creating Event Handlers in XAML

Expression Blend 2 August Preview does not currently have a user interface to set event handlers, so we will manually set them in **Scene.xaml** (Fig. 19.12). The **timelineTimer** **Storyboard**'s **Completed** attribute (line 8) registers an event that calls the **updateTime** function located in our JavaScript code-behind file (Fig. 19.13) when the animation has completed. This JavaScript function updates user-interface elements such as the timeline marker.

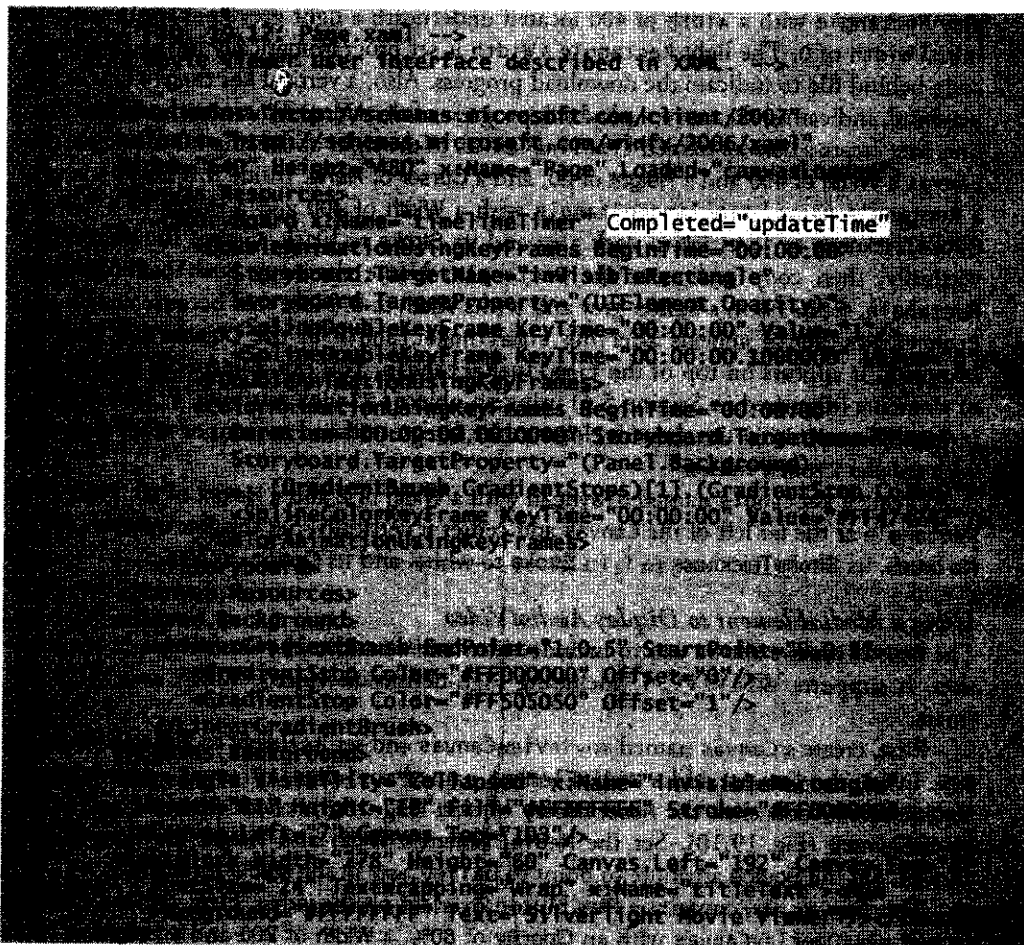


Fig. 19.12 | Movie Viewer user interface described in XAML. (Part 1 of 5.)

```

28 <Canvas x:Name="controls" Width="640" Height="160" Canvas.Top="320"
29 <Canvas MouseLeftButtonDown="movieThumbHandler" Cursor="Hand"
30 Width="120" Height="114" Canvas.Left="33" Canvas.Top="38"
31 x:Name="crazyDogButton">
32 <Image Width="120" Height="90" Source="batterThumb.jpg" />
33 <TextBlock Width="78" Height="24" Canvas.Left="23"
34 Canvas.Top="90" TextWrapping="Wrap"><Run
35 Foreground="FFFFFFFF" Text="Crazy Dog" /></TextBlock>
36 </Canvas>
37 <Canvas MouseLeftButtonDown="movieThumbHandler" Width="120"
38 Height="114" Canvas.Left="154" Canvas.Top="38" Cursor="Hand"
39 x:Name="gravityButton">
40 <Image Width="120" Height="90"
41 Source="featherAndHamerThumb.jpg" />
42 <TextBlock Width="52" Height="24" Canvas.Top="90"
43 TextWrapping="Wrap" Canvas.Left="34"><Run
44 Foreground="FFFFFFFF" Text="Gravity" /></TextBlock>
45 </Canvas>
46 <Canvas MouseLeftButtonDown="movieThumbHandler" Width="120"
47 Height="114" Canvas.Left="335" Canvas.Top="38" Cursor="Hand"
48 x:Name="apolloButton">
49 <Image Width="120" Height="90" Source="apollo13LaunchPad.jpg" />
50 <TextBlock Width="72" Height="24" Canvas.Left="26"
51 Canvas.Top="90" TextWrapping="Wrap">
52 <Run Foreground="FFFFFFFF" Text="Apollo 13" /></TextBlock>
53 </Canvas>
54 <Canvas MouseLeftButtonDown="movieThumbHandler" Width="120"
55 Height="114" Canvas.Left="487" Canvas.Top="38" Cursor="Hand"
56 x:Name="f35Button">
57 <Image Width="120" Height="90" Source="F35Thumb.jpg" />
58 <TextBlock Width="88" Height="24" Canvas.Left="18"
59 Canvas.Top="90" TextWrapping="Wrap"><Run
60 Foreground="FFFFFFFF" Text="F35 Landing" /></TextBlock>
61 </Canvas>
62
63 <!-- Define the buttons -->
64 <Canvas MouseLeftButtonDown="playAndPauseButtonEventHandler"
65 Width="30" Height="30" x:Name="playButton" Cursor="Hand"
66 Canvas.Left="10">
67 <Rectangle Stroke="FF000000" Width="30" Height="30"
68 RadiusX="1" RadiusY="1">
69 <Rectangle.Fill>
70 <LinearGradientBrush EndPoint="1,0.5" StartPoint="0,0">
71 <GradientStop Color="FF0000FF" Offset="0.5" />
72 <GradientStop Color="FF0000FF" Offset="1" />
73 </LinearGradientBrush>
74 </Rectangle.Fill>
75 </Rectangle>
76 </Canvas>
77 <Canvas MouseLeftButtonDown="playAndPauseButtonEventHandler"
78 Width="30" Height="30" x:Name="pauseButton" Cursor="Hand"
79 Canvas.Left="40">
80 <Rectangle Stroke="FF000000" Width="30" Height="30"
81 RadiusX="1" RadiusY="1">
82 <Rectangle.Fill>
83 <LinearGradientBrush EndPoint="1,0.5" StartPoint="0,0">
84 <GradientStop Color="FF0000FF" Offset="0.5" />
85 <GradientStop Color="FF0000FF" Offset="1" />
86 </LinearGradientBrush>
87 </Rectangle.Fill>
88 </Rectangle>
89 </Canvas>

```

Fig. 19.12 | Movie Viewer user interface described in XAML. (Part 2 of 5.)

```

88     <Path Stretch="Fill" Stroke="#FFFFFF"
89           StrokeThickness="4" Width="12" Height="12" RenderTransformOrigin="0,0.5"
90           L244,403" RenderTransformOrigin="0,0.5"
91           StrokeEndLineCap="Square" StrokeStartLineCap="Round"
92           Canvas.Left="10" Canvas.Top="11">
93     <Path.RenderTransform>
94       <TransformGroup>
95         <ScaleTransform ScaleX="1" ScaleY="1"/>
96         <SkewTransform AngleX="0" AngleY="0"/>
97         <RotateTransform Angle="0"/>
98         <TranslateTransform X="0" Y="0"/>
99       </TransformGroup>
100     </Path.RenderTransform>
101 </Path>
102 </Canvas>
103 <Canvas x:Name="timeCanvas" Width="75" Height="23" Canvas.Left="10"
104         Canvas.Top="3">
105   <Rectangle Stroke="#FF000000" Width="75" Height="23" RadiusX="2"
106           RadiusY="2" StrokeThickness="1">
107     <Rectangle.Fill>
108       <LinearGradientBrush EndPoint="1,0.5" StartPoint="0,0.5"
109           GradientStop Color="#FF8B8B" Offset="0"/>
110       <GradientStop Color="#FFFFFF" Offset="1"/>
111     </LinearGradientBrush>
112   </Rectangle.Fill>
113 </Rectangle>
114   <TextBlock x:Name="timeText" Width="75" Height="23"
115           Foreground="#FF000000" TextWrapping="Wrap" Canvas.Left="10"
116           Canvas.Top="3"><Run Text="00:00:00"/></TextBlock>
117 </Canvas>
118 <Canvas MouseLeftButtonDown="volumeHandler" Canvas.Left="10"
119         x:Name="volumeCanvas" Width="15" Height="20"
120         Canvas.Left="616">
121   <Rectangle Fill="#FF868686"
122           Stroke="#FF000000" Width="15" Height="20" Canvas.Left="616"
123           Canvas.Top="14">
124     <Rectangle Opacity="0.5" x:Name="volumeHead" Width="15"
125           Height="2" Fill="#FFFFFF" Stroke="#FFFFFF"
126           StrokeThickness="0" RadiusX="0" RadiusY="0" Canvas.Left="616"
127           Canvas.Top="14"/>
128 </Canvas>
129 <Canvas x:Name="timelineCanvas" Width="403" Height="11" Canvas.Left="87"
130         Canvas.Top="2" Cursor="Hand">
131   <Rectangle x:Name="timelineRectangle" Width="403" Height="11"
132           Fill="#FFA9A9A9" Stroke="#FF000000" Canvas.Left="87"
133           Canvas.Top="2">
134     <Rectangle MouseLeftButtonDown="timelineHandler"
135           x:Name="downloadProgressRectangle" Width="0"
136           Height="4" Fill="#FFD5D5D5" Stroke="#FF000000"
137           Canvas.Left="87" Canvas.Top="11"/>
138     <Rectangle Opacity="0.5" x:Name="playhead" Width="15" Height="2"
139           Stroke="#FFFFFF" Canvas.Left="11" Canvas.Top="3"/>
140 </Canvas>
141 <Canvas MouseLeftButtonDown="playAndPauseButtonEventHandler"
142         Width="30" Height="30" x:Name="playbutton" Canvas.Left="10"

```

Fig. 19.12 | Movie Viewer user interface described in XAML. (Part 3 of 5.)

```

147 Canvas.Left="10" Visibility="Collapsed" >
148   <Rectangle Stroke="#FF00000" Width="30" Height="30"
149     RadiusX="4" RadiusY="4">
150     <Rectangle.Fill>
151       <LinearGradientBrush EndPoint="1,0.5" StartPoint="0,0"
152         MappingMode="RelativeToBoundingBox" SpreadMethod="Pad"
153         <GradientStop Color="#FF0000FF" Offset="0"/>
154         <GradientStop Color="#FF0084FF" Offset="1"/>
155       </LinearGradientBrush>
156     </Rectangle.Fill>
157   </Rectangle>
158   <Canvas Width="Stretch" Height="Stretch" Stroke="#FFFFFFFF"
159     StrokeThickness="3" Width="4" Height="18"
160     RenderTransformOrigin="0.5,0.5" StrokeEndLineCap="Round"
161     StrokeStartLineCap="Round" Canvas.Left="9"
162     StrokeDashCap="Flat" Canvas.Top="6" Data="M223,388L223,300"/>
163   <Path Stretch="Fill" Stroke="#FFFFFFFF"
164     StrokeThickness="4" Width="4" Height="16"
165     Data="M223,388L223,403" RenderTransformOrigin="0.5,0.5"
166     StrokeEndLineCap="Round" StrokeStartLineCap="Round"
167     Canvas.Left="17" StrokeDashCap="Flat" Canvas.Top="6"/>
168 </Canvas>
169 <Canvas MouseLeftButtonDown="toggleFullScreen" Width="30"
170   Height="30" x:Name="fullScreenButton" Cursor="Hand"
171   Canvas.Left="582" Canvas.Top="3">
172   <Rectangle Stroke="#FF00000" Width="30" Height="30"
173     RadiusX="4" RadiusY="4">
174     <Rectangle.Fill>
175       <LinearGradientBrush EndPoint="1,0.5" StartPoint="0,0"
176         MappingMode="RelativeToBoundingBox" SpreadMethod="Pad"
177         <GradientStop Color="#FF0000FF" Offset="0"/>
178         <GradientStop Color="#FF0084FF" Offset="1"/>
179       </LinearGradientBrush>
180     </Rectangle.Fill>
181   </Rectangle>
182   <Rectangle Width="22" Height="22" Stroke="#FFFFFFFF"
183     StrokeEndLineCap="Square" StrokeStartLineCap="Round"
184     StrokeThickness="2" RadiusX="2" RadiusY="2" Canvas.Left="4"
185     Canvas.Top="4"/>
186   <Rectangle Width="10" Height="10" Stroke="#FFFFFFFF"
187     StrokeEndLineCap="Square" StrokeStartLineCap="Round"
188     StrokeThickness="2" RadiusX="2" RadiusY="2" Canvas.Left="4"
189     Canvas.Top="16"/>
190 </Canvas>
191 <Canvas MouseLeftButtonDown="stopButtonEventHandler" Width="30"
192   Height="30" x:Name="stopButton" Cursor="Hand" Canvas.Left="582"
193   <Rectangle Stroke="#FF00000" Width="30" Height="30"
194     RadiusX="4" RadiusY="4">
195     <Rectangle.Fill>
196       <LinearGradientBrush EndPoint="1,0.5" StartPoint="0,0"
197         MappingMode="RelativeToBoundingBox" SpreadMethod="Pad"
198         <GradientStop Color="#FF0000FF" Offset="0"/>
199         <GradientStop Color="#FF0084FF" Offset="1"/>
200       </LinearGradientBrush>

```

Fig. 19.12 | Movie Viewer user interface described in XAML. (Part 4 of 5.)

```

298         </LinearGradientBrush>
299     </Rectangle.Fill>
300 </Rectangle>
301 <Rectangle RadiusX="2" RadiusY="2" Width="14"
302     Height="18" Canvas.Left="8" Canvas.Top="6"
303     StrokeThickness="0" Canvas.Zip="0"/>
304 </Canvas>
305 </Canvas>
306 <Canvas x:Name="movieViewCanvas" Width="640" Height="260"
307     Canvas.Top="46">
308     <MediaElement AutoPlay="false" MediaEnded="movieEndedHandler"
309         MediaOpened="movieOpenedHandler" x:Name="movieMediaElement"
310         Width="640" Height="260" Source="Ball.mp4"/>
311     <Canvas MouseLeftButtonDown="playAndPauseButtonEventHandler"
312         Width="200" Height="180" Canvas.Left="200" Canvas.Top="
313         Opacity="0.6" Visibility="Collapsed" x:Name="playAndPauseButton">
314         <Canvas.RenderTransform>
315             <TransformGroup>
316                 <ScaleTransform ScaleX="1" ScaleY="1"/>
317                 <SkewTransform AngleX="0" AngleY="0"/>
318                 <RotateTransform Angle="0"/>
319                 <TranslateTransform X="0" Y="0"/>
320             </TransformGroup>
321         </Canvas.RenderTransform>
322         <Rectangle Fill="#FF000000"
323             Width="200" Height="180" RadiusX="40"
324             RadiusY="40" Canvas.Left="0" Canvas.Top="0"/>
325         <Ellipse Stroke="#FFFFFF" StrokeThickness="6"
326             Width="100" Height="100" Canvas.Left="40" Canvas.Top="40"
327             Path.Fill="#FFD00000" Stretch="Fill" StrokeDashOffset="0"
328             StrokeThickness="6" Width="30" Height="30" Canvas.Left="
329             Canvas.Top="30" Data="M95,189 L119,113"
330             StrokeEndLineCap="Round" StrokeDashOffset="0"
331             Path.Fill="#FF010000" Stretch="Fill" StrokeDashOffset="0"
332             StrokeThickness="6" Width="30" Height="30" Canvas.Left="
333             Canvas.Top="61" Data="M189,189 L213,213"
334             RenderTransformOrigin="0.5,0.5" StrokeEndLineCap="Round"
335             StrokeStartLineCap="Round"/>
336         <Path.RenderTransform>
337             <TransformGroup>
338                 <ScaleTransform ScaleX="1" ScaleY="1"/>
339                 <SkewTransform AngleX="0" AngleY="0"/>
340                 <RotateTransform Angle="0"/>
341                 <TranslateTransform X="0" Y="0"/>
342             </TransformGroup>
343         </Path.RenderTransform>
344     </Canvas>
345 </Canvas>
346 <TextBlock Width="70" Height="30" Canvas.Left="200"
347     Canvas.Top="120" FontSize="30" FontFamily="Serif"
348     Text="Play" ToolTip="Play" Width="70"
349 </TextBlock>
350 </Canvas>
351 </Canvas>

```

Fig. 19.12 | Movie Viewer user interface described in XAML. (Part 5 of 5.)

Configuring the Event Handlers

For each of the thumbnail button Canvases (crazyDogButton, gravityButton, apo11b15Button and f358Button), we specify a **MouseLeftButtonDown** attribute (lines 36, 44, 53 and 61, respectively). This registers the `movieThumbHandler` function (Fig. 19.13, lines 157–178) as the event handler to call when the user clicks one of these Canvases with the left mouse button. Each of the playback control buttons also has a **MouseLeftButtonDown** attribute (lines 71, 139, 163, 185 and 207 for the play, pause, full-screen, stop, play overlay buttons, respectively). Each of these buttons has a separate event handler function.

The `volumeCanvas` has a **MouseLeftButtonDown** attribute (line 118) that allows the user to change the volume by calling `volumeHandler` (Fig. 19.13, lines 239–245) when the user clicks somewhere on the `volumeCanvas`.

The `downloadProgressRectangle` has a **MouseLeftButtonDown** attribute (line 132) that allows the user to jump anywhere in the video by calling the `timelineHandler` function (lines 225–236, Fig. 19.13) when the user clicks somewhere on the `downloadProgressRectangle`.

The `movieMediaElement`'s **MediaOpened** attribute (line 205) is set to `movieOpenedHandler`. When a new video is opened, function `movieOpenedHandler` (Fig. 19.13, lines 137–143) is called to ensure that the **Play** overlay button is visible, and to start the timer that keeps the timeline and time up to date. When you open a movie the `MediaElement` begins playing the movie by default. We don't want this to happen until the user clicks the play button, so we set its **AutoPlay** attribute to `false` (line 204). The `movieMediaElement`'s **MediaEnded** attribute (line 204) is set to `movieEndedHandler`. When the video finishes playing, the `movieEndedHandler` function (Fig. 19.13, lines 146–155) is called to reset the video to the beginning and to ensure that the **Play** overlay button is visible.

Registering Event Handlers in JavaScript

An alternative to registering event handlers in the XAML is to register event handlers in the JavaScript code. While this technique requires a few more lines of code, it has two key advantages. First, it keeps the application's logic (in this case, event handling) separate from the application's user interface. Second, it allows you to add and remove event listeners dynamically. The JavaScript for adding an event handler is:

```
variableName = objectName.addEventHandler( "EventName", eventHandler );
```

The JavaScript for removing an event handler is:

```
objectName.removeEventHandler( "EventName", variableName );
```

When an event is registered in JavaScript using the **addEventListener** method, we must assign the return value of the method to a variable. This way, if we wish to remove an event listener using the **removeEventListener**, we can remove only that specific event listener.

19.4.4 Using JavaScript for Event Handling and DOM Manipulation

The JavaScript code-behind file, `Page.xaml.js` (Fig. 19.13), defines the event handlers for the various elements in the XAML. In the event handlers, we use JavaScript to manipulate the Silverlight DOM, similar to how we manipulated the XHTML DOM in Chapter 12 and the XML DOM in Chapter 14. To edit the JavaScript code files, use your preferred text editor.

```

1 // Fig. 19.13: Page.xaml.js
2 // JavaScript code behind for Movie Viewer.
3
4 // variables for accessing the Silverlight elements.
5 var host; // allow access to host plug-in
6 var Page;
7 var movieMediaElement;
8 var downloadProgressRectangle;
9 var timeText;
10 var timelineRectangle;
11 var playhead;
12 var timelineTimer;
13 var playButton;
14 var pauseButton;
15 var playOverlayCanvas;
16 var timelineTimer;
17 var volumeCanvas;
18 var volumeHead;
19 var crazyDogButton;
20 var gravityButton;
21 var apollo15Button;
22 var f35Button;
23 var controls;
24 var fullscreenButton;
25 var timeCanvas;
26 var titleText;
27 var playOverlayCanvasListener; // token for event listener
28
29 function canvasLoaded( sender, EventArgs )
30 {
31     // set variables to more easily access the Silverlight elements
32     host = sender.getHost(); // allow access to host plug-in
33     Page = sender.findName( "Page" );
34     movieMediaElement = sender.findName( "movieMediaElement" );
35     downloadProgressRectangle = sender.findName(
36         "downloadProgressRectangle" );
37     timeText = sender.findName( "timeText" );
38     timelineRectangle = sender.findName( "timelineRectangle" );
39     playhead = sender.findName( "playhead" );
40     timelineTimer = sender.findName( "timelineTimer" );
41     playButton = sender.findName( "playButton" );
42     pauseButton = sender.findName( "pauseButton" );
43     playOverlayCanvas = sender.findName( "playOverlayCanvas" );
44     volumeCanvas = sender.findName( "volumeCanvas" );
45     volumeHead = sender.findName( "volumeHead" );
46     crazyDogButton = sender.findName( "crazyDogButton" );
47     gravityButton = sender.findName( "gravityButton" );
48     apollo15Button = sender.findName( "apollo15Button" );
49     f35Button = sender.findName( "f35Button" );
50     controls = sender.findName( "controls" );
51     fullscreenButton = sender.findName( "fullscreenButton" );
52     timeCanvas = sender.findName( "timeCanvas" );
53     titleText = sender.findName( "titleText" );

```

Fig. 19.13 | JavaScript code-behind file for Movie Viewer. (Part 1 of 6.)

```

61 // set an event handler for the onFullScreenChange event
62 host.content.onFullScreenChange = onFullScreenChangedHandler;
63
64 // start the timer
65 timelineTimer.begin();
66
67 // add function canvasLoaded
68
69 // this is the event handler
70 updateTimeline()
71
72 // get the video's current position in seconds
73 var elapsedTime = movieMediaElement.position.Seconds;
74 var hours = conversionHMSS( elapsedTime )[ 0 ]; // saves hours
75 var minutes = conversionHMSS( elapsedTime )[ 1 ]; // saves minutes
76 var seconds = conversionHMSS( elapsedTime )[ 2 ]; // saves seconds
77
78 // set text of timeText to current time in hh:mm:ss format
79 timeText.text = hours + ":" + minutes + ":" + seconds;
80
81 // set width of downloadProgressRectangle
82 downloadProgressRectangle.width = movieMediaElement.downloadProgress *
83 timelineRectangle.width;
84
85 // if the movie is playing, place the playhead at a
86 // position representing the playback progress
87 ( movieMediaElement.position.Seconds &&
88 movieMediaElement.naturalDuration )
89
90 playHead[ "Canvas.Left" ] = ( ( movieMediaElement.position.Seconds /
91 movieMediaElement.naturalDuration.Seconds ) *
92 timelineRectangle.Width ) + timelineRectangle[ "Canvas.Left" ];
93
94 // if the movie is not playing, place the playhead at the beginning
95 playHead[ "Canvas.Left" ] = timelineRectangle[ "Canvas.Left" ];
96
97 // if the video is incomplete or movie is playing
98 ( movieMediaElement.downloadProgress != 1 ||
99 movieMediaElement.currentState == "Playing" )
100
101 timelineTimer.begin(); // run timelineTimer again
102
103 // function updateTime
104
105 // enable play and pause buttons
106 playAndPauseButtonEventHandler( sender, eventArgs )
107
108 // check the CurrentState of the movie;
109 // pause if playing, play if paused or stopped

```

Fig. 19.13 | JavaScript code-behind file for Movie Viewer. (Part 2 of 6.)

```

107 if ( movieMediaElement.CurrentState == "Playing" )
108 {
109     movieMediaElement.pause();
110
111     playButton.Visibility = "Visible"; // show play button
112     pauseButton.Visibility = "Collapsed"; // hide pause button
113 } // end if
114 else
115 {
116     movieMediaElement.play();
117     timelineTimer.begin(); // start timelineTimer again
118     pauseButton.Visibility = "Visible"; // show pause button
119     playButton.Visibility = "Collapsed"; // hide play button
120     playOverlayCanvas.Visibility = "Collapsed"; // hide "Play" overlay
121 } // end if
122 // and function playAndPauseButtonEventHandler
123
124 // handle stop button
125 function stopButtonEventHandler( sender, eventArgs )
126 {
127     movieMediaElement.stop(); // stop the movie
128     playButton.Visibility = "Visible"; // show play button
129     pauseButton.Visibility = "Collapsed"; // hide pause button
130
131     // show "Play" overlay
132     playOverlayCanvas.Visibility = "Visible";
133     updateTime();
134 } // and function stopButtonEventHandler
135
136 // handle MediaOpened event
137 function movieOpenedHandler( sender, eventArgs )
138 {
139     timelineTimer.begin();
140
141     // show "Play" overlay
142     playOverlayCanvas.Visibility = "Visible";
143 } // and function movieOpenedHandler
144
145 // handle when movie has reached end
146 function movieEndedHandler( sender, eventArgs )
147 {
148     movieMediaElement.stop(); // stop the movie
149     playButton.Visibility = "Visible"; // show play button
150     pauseButton.Visibility = "Collapsed"; // hide pause button
151
152     // show "Play" overlay
153     playOverlayCanvas.Visibility = "Visible";
154     updateTime();
155 } // and function movieEndedHandler
156
157 function movieThumbHandler ( sender, eventArgs ) // a thumb
158 {
159     movieMediaElement.stop(); // stop movie

```

Fig. 19.13 | JavaScript code-behind file for Movie Viewer. (Part 3 of 6.)

```

160 playButton.Visibility = "Visible"; // show play button
161 pauseButton.Visibility = "Collapsed"; // hide pause button
162
163 switch ( sender.name )
164 {
165     case "crazyDogButton": // open Crazy Dog video
166         movieMediaElement.source = "bailey.wmv";
167         break;
168     case "gravityButton": // open Gravity video
169         movieMediaElement.source = "featherAndHammer.wmv";
170         break;
171     case "apollo15Button": // open Apollo 15 video
172         movieMediaElement.source = "apollo15Launch.wmv";
173         break;
174     case "f35Button": // open F35 Landing video
175         movieMediaElement.source = "F35.wmv";
176         break;
177 } // end switch
178 } // end function movieThumbHandler
179
180 // handle toggle full-screen button by toggling fullScreen state
181 function toggleFullScreen( sender, eventArgs )
182 {
183     host.content.fullScreen = !host.content.fullScreen;
184 } // end function toggleFullScreen
185
186 // handle onFullScreenChange event
187 function onFullScreenChangedHandler( sender, eventArgs )
188 {
189     // update layout based on current dimensions
190     updateLayout( host.content.actualWidth,
191                 host.content.actualHeight );
192
193     // update time and timeline
194     updateTime();
195 } // end function onFullScreenChangedHandler
196
197 // reposition and resize elements based on new dimensions
198 function updateLayout( width, height )
199 {
200     // resize and reposition the elements based on the screen dimensions
201     pageWidth = width;
202     pageHeight = height;
203     movieMediaElement.width = width;
204     movieMediaElement.height = height - 220;
205     movieMediaElement[ "Canvas.Left" ] =
206         ( width / 2 ) - ( ( movieMediaElement.width ) / 2 );
207     movieMediaElement[ "Canvas.Top" ] =
208         ( ( height - 220 ) / 2 ) - ( movieMediaElement.height / 2 );
209     controlsWidth = width;
210     playOverlayCanvas[ "Canvas.Left" ] =
211         ( width / 2 ) - ( ( playOverlayCanvas.width ) / 2 );

```

Fig. 19.13 | JavaScript code-behind file for Movie Viewer. (Part 4 of 6.)

```

212 playOverlayCanvas[ "Canvas.Top" ] =
213     ( ( height - 220 ) / 2 ) - ( playOverlayCanvas.height / 2 );
214 controls[ "Canvas.Left" ] = ( width / 2 ) - ( ( controls.width ) / 2 );
215 controls[ "Canvas.Top" ] = height - controls.height;
216 timelineRectangle.width = controls.width - 235;
217 fullscreenButton[ "Canvas.Left" ] = controls.width - 55;
218 timeCanvas[ "Canvas.Left" ] = controls.width - 140;
219 volumeCanvas[ "Canvas.Left" ] = controls.width - 22;
220 titleText[ "Canvas.Left" ] =
221     ( width / 2 ) - ( ( titleText.width ) / 2 );
222 } // end function updateLayout
223
224 // handle timelineCanvas's MouseLeftButtonDown event
225 function timelineHandler( sender, eventArgs )
226 {
227     // determine new time from mouse position
228     var elapsedTime = ( ( eventArgs.getPosition( timelineRectangle ).x ) /
229         timelineRectangle.Width ) *
230         movieMediaElement.NaturalDuration.seconds;
231     var hours = convertToHHMMSS( elapsedTime )[ 0 ]; // Saves hours
232     var minutes = convertToHHMMSS( elapsedTime )[ 1 ]; // Saves minutes
233     var seconds = convertToHHMMSS( elapsedTime )[ 2 ]; // Saves seconds
234     movieMediaElement.Position = hours + ":" + minutes + ":" + seconds;
235     updateTime();
236 } // end function timelineHandler
237
238 // handle volume's MouseLeftButtonDown event
239 function volumeHandler( sender, eventArgs )
240 {
241     movieMediaElement.volume = 1 - ( ( eventArgs.getPosition(
242         volumeCanvas ).y ) / 30 );
243     volumeHead[ "Canvas.Top" ] =
244         eventArgs.getPosition( volumeCanvas ).y;
245 } // end function volumeHandler
246
247 // get the hours, minutes and seconds of the video's current position
248 // Date object converts seconds to hh:mm:ss format
249 function convertToHHMMSS( seconds )
250 {
251     var datetime = new Date( 0, 0, 0, 0, 0, seconds );
252     var hours = datetime.getHours(); // saves hours to var
253     var minutes = datetime.getMinutes(); // saves minutes to var
254     var seconds = datetime.getSeconds(); // saves seconds to var
255
256     // ensure hh:mm:ss format
257     if ( seconds < 10 )
258     {
259         seconds = "0" + seconds;
260     } // end if
261
262     if ( minutes < 10 )
263     {

```

Fig. 19.13 | JavaScript code-behind file for Movie Viewer. (Part 5 of 6.)

```

246         minutes = "0" + minutes;
247     }
248     // and seconds
249     seconds = seconds % 60;
250     // and minutes
251     minutes = minutes % 60;
252     // and hours
253     hours = hours % 24;
254     // and if
255     // return | hours, minutes, seconds |
256     return [hours, minutes, seconds];
257 }

```

Fig. 19.13 | JavaScript code-behind file for Movie Viewer. (Part 6 of 6.)

Handling Events and Accessing XAML Elements in JavaScript

Lines 5–27 declare variables that our event handler functions use to access the XAML elements in our video player. In the `canvasLoaded` function (lines 29–60), which handles the Page Canvas's **Loaded** event (Fig. 19.12, line 6), these variables are set to reference their corresponding XAML elements using the sender's `findName` method (lines 33–53). Every event handler receives `sender` and `eventArgs` parameters. The `sender` parameter is a reference to the element with which the user interacted, and the `eventArgs` parameter passes information about the event that occurred. Line 32 sets the `host` variable to the Silverlight plug-in object using the `getHost` method. This allows us to access properties of the Silverlight plug-in, such as its screen dimensions, throughout the program. Line 56 registers an event handler for the plug-in's `onFullScreenChange` event. Line 59 calls the `timelineTimer` Storyboard's `begin` function, to start the Storyboard that we are using as a timer. When this Storyboard's `Completed` event (Fig. 19.12, line 8) occurs—i.e., its 0.1-second-long animation completes (Fig. 19.12, lines 9–14)—the event handler `updateTime` (lines 63–100) is invoked.

Creating a Timer

The `updateTime` function (lines 63–100) updates the `timeText`, the `downloadProgressRectangle`, the `playHead`, and starts the `timelineTimer` again if necessary. It uses the `convertToHHMMSS` function (lines 249–273) to convert the `movieMediaElement`'s `position.Seconds`—its elapsed time in seconds—to hours, minutes and seconds (lines 66–69), then displays that time in `hh:mm:ss` format in the `timeText` textBlock (line 72). The `updateTime` function also updates the download progress indicator (lines 75–76) by setting the width of the `downloadProgressRectangle` to the width of the `timelineRectangle` multiplied by the `movieMediaElement`'s `downloadProgress`, which is a value from 0 to 1 representing the fraction of the video that has downloaded so far. Lines 80–81 check whether `movieMediaElement`'s `naturalDuration` and `position.Seconds` properties exist. If they do, lines 83–85 set the `playHead`'s position to the current playback position in the video. This is accomplished by setting the `playHead`'s `Canvas.Left` attribute to the sum of the `timelineRectangle`'s `Canvas.Left` attribute and the width of the `timelineRectangle` multiplied by the ratio of current time (`movieMediaElement.position.Seconds`) and total time (`movieMediaElement.naturalDuration.Seconds`). `Left` is a **dependency property** of `Canvas`, meaning that the `Left` value is relative to that of the `Canvas`. Since the `Canvas.Left` dependency property already has a dot in its notation, we must enclose the attribute name in quotes and square brackets, as in `element["attributeName"]`. If

`movieMediaElement`'s `naturalDuration` and `positionSeconds` attributes do not exist, line 91 sets the `playHead`'s `Canvas.Left` attribute to be equal to the `timelineRectangle`'s to indicate that the movie has not started playing. Finally, lines 95–99 check whether the download is not finished or the movie is playing, in which case it calls `timelineTimer`'s `begin` function to run the timer again. This way, the `downloadProgressRectangle` and `playHead` will be updated.

Handling Button Events

The `playAndPauseButtonEventHandler` function (lines 103–122) handles the play and pause buttons' `MouseDown` events (Fig. 19.12, lines 71 and 139). Line 107 checks whether the `movieMediaElement` is currently playing. If it is, the video should be paused and the play button should be shown. If not, lines 116–120 play the video, start the `timelineTimer`, show the pause button and hide the `playOverlayCanvas`.

The `stopButtonEventHandler` function (lines 125–134) handles the stop button. It stops the video (line 127), then shows the play button and the **Play** overlay button (lines 128–132). Finally, it calls the `updateTime` function to ensure that the `timeText` `TextBlock` displays `00:00:00`.

Function `movieThumbHandler` (lines 157–178) handles the movie thumbnail buttons. Lines 159–161 stop the video and show the play button. Lines 163–177 contain a `switch` statement that checks the sender element's `name` attribute (the name of the button that was clicked) and sets the `Source` of `movieMediaElement` to the corresponding video file.

Adding a Full-Screen Feature

Function `toggleFullScreen` (lines 181–184) handles the full-screen button. Line 183 sets the Silverlight plug-in's `fullScreen` attribute to the opposite of its previous value. This means that if the plug-in was previously in full-screen mode, it will switch to windowed mode, and if it was previously in windowed mode, it will switch to full-screen mode.

The `onFullScreenChangeHandler` function (lines 208–216) handles the `onFullScreenChange` event. Lines 190–191 uses the `updateLayout` function to update the layout of the application based on its current width and height. Then, line 194 calls the `updateTime` function.

Dynamically Changing XAML Layout

Function `updateLayout` (lines 198–222) repositions the user interface elements relative to the width and height parameters. Lines 201–202 set the width and height of the `PageCanvas` to the width and height parameters. Lines 203–204 set the width and height of `movieMediaElement` to the width parameter and the height parameter minus 220 (leaving room for the controls). Lines 205–208 move the `movieMediaElement` in the application. Line 209 sets the width of the `controlsCanvas`. Lines 210–213 center the `playOverlayCanvas` over the `movieMediaElement`. Lines 214–215 center the controls at the bottom of the application. Line 216 sets the width of the `timelineRectangle` to be the width of the `controlsCanvas` minus 235, to allow room for the other controls. Lines 217–219 move the full-screen button, `timeCanvas` and `volumeCanvas` to the right side of the `controlsCanvas`. Finally, lines 220–221 center `titleText` at the top of the application.

The `movieOpenedHandler` function (lines 137–143) handles `movieMediaElement`'s `MediaOpened` event (Fig. 19.12, line 205). Line 139 starts the `timelineTimer` to update the loading progress rectangle, and line 142 shows the `playOverlayCanvas`.

The `movieEndedHandler` function (lines 146–155) handles the `movieMediaElement`'s `MediaEnded` event (Fig. 19.12, line 204). It stops the video (line 148), which resets the playback position to the beginning, then shows the play button and hides the pause button (lines 149–150). Lines 153 shows the `playOverlayCanvas`. Finally, line 154 calls the `updateTime` function to ensure that the `timeText` `textBox` displays `00:00:00`.

Creating a Timeline for a MediaElement

The `timelineHandler` function (lines 225–236) handles the `downloadProgressRectangle`'s `MouseDown` event (Fig. 19.12, line 132). Lines 228–234 set `elapsedTime` to the position that was clicked on the timeline, convert the number of seconds to `hh:mm:ss` format using the `convertToHHMMSS` function and set the `Position` of the `movieMediaElement` to that time string. Finally, line 235 calls the `updateTime` function to show the new position.

Controlling Volume of a MediaElement

Function `volumeHandler` (lines 239–245) handles the `volumeCanvas`'s `MouseDown` event (Fig. 19.12, line 118). Lines 241–242 set `movieMediaElement`'s `volume` property based on the position the user clicked on the `volumeCanvas`. We convert the `y`-coordinate of the mouse relative to the volume rectangle to a value between 0 and 1 (0 being muted and 1 being full volume). Lines 243–244 move the `volumeHead` to the new position.

19.5 Embedding Silverlight in HTML

Expression Blend generates an HTML wrapper named `Default.html` for your Silverlight application when you first create the Silverlight project. Figure 19.14 shows a version of this file that we formatted for readability. You can open `Default.html` in a supported web browser to test your application. You can embed a Silverlight application into an existing HTML file by including the scripts (lines 8–10), the `silverlightHost` style class (lines 12–13) and the `SilverlightControlHost` div (lines 17–21). You can adjust the width and height of your application by changing the width and height attributes of the `silverlightHost` style class (lines 12–13).

```

1 <!-- Fig. 19.14: Default.html -->
2 <!-- HTML wrapper for Movie Viewer. -->
3 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
4   "http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
5 <html xmlns = "http://www.w3.org/1999/xhtml">
6   <head>
7     <title>MovieViewer</title>
8     <script type = "text/javascript" src = "Silverlight.js"></script>
9     <script type = "text/javascript" src = "Default_html.js"></script>
10    <script type = "text/javascript" src = "Page.xaml.js"></script>
11    <style type = "text/css">
12      .silverlightHost { height: 480px;
13                        width: 640px }
14    </style>
15  </head>

```

Fig. 19.14 | HTML wrapper for Movie Viewer. (Part 1 of 2.)

```

16 <body>
17   <div id = "SilverlightControlHost" class = "silverlightHost">
18     <script type = "text/javascript">
19       createSilverlight();
20     </script>
21   </div>
22 </body>
23 </html>

```

Fig. 19.14 | HTML wrapper for Movie Viewer. (Part 2 of 2.)

The `createSilverlight` function (line 19) is located in `Default_html.js` (Fig. 19.15). This function inserts the Silverlight plug-in object in the `SilverlightControlHost` div. The `Default_html.js` file that Expression Blend creates will not work with our project because it tries to access function `Page` in the JavaScript, which no longer exists. You must remove the lines instantiating the `scene` variable and set the `onLoad` event to `null` (line 15).

```

1 // Fig. 19.15: Default_html.js
2 // Create Silverlight object in SilverlightControlHost div.
3 function createSilverlight()
4 {
5   Silverlight.createObjectEx( {
6     source: "Page.xaml",
7     parentElement: document.getElementById( "SilverlightControlHost" ),
8     id: "SilverlightControl",
9     properties: {
10      width: "100%",
11      height: "100%",
12      version: "1.0"
13    },
14    events: {
15      onLoad: null
16    }
17  } );
18 }
19
20 if ( !window.Silverlight )
21   window.Silverlight = {};
22
23 Silverlight.createDelegate = function( instance, method ) {
24   return function() {
25     return method.apply( instance, arguments );
26   }
27 }

```

Fig. 19.15 | Creates Silverlight object in `SilverlightControlHost` div.

19.6 Silverlight Streaming

Microsoft provides a service called **Silverlight Streaming** at `silverlight.live.com`. This service currently hosts your Silverlight applications for free, which allows individuals and

businesses to share video content online without having to provide and pay for the significant bandwidth that video requires. While in prerelease status, Silverlight Streaming provides you with “up to 4GB storage and unlimited outbound streaming, and no limit on the number of users that can view those streams.”¹ Eventually, Microsoft intends to allow “up to 1 million minutes of free video streaming at 700 Kbps per site per month. Unlimited streaming will also be available for free with advertising, or with payment of a nominal fee for the service for use without advertising.”² You can easily embed Silverlight applications that are hosted by this service into your web pages.

Encoding Your Video with Expression Media Encoder

According to dev.live.com/silverlight/, the bit rate of video files included with Silverlight applications must not exceed 700 Kbps. To ensure that your video adheres to these requirements, it is recommended that you encode your video using Microsoft Expression Blend Media Encoder. A free trial of Media Encoder is available at www.microsoft.com/expression/products/download.aspx?key=encoder. Once Media Encoder is installed, open it and select **Import...** from the **File** menu. Select the video file you would like to encode (Media Encoder supports many video types) and click **Open**. If the video is encoded in VC-1 and doesn't open properly, you may need to install Windows Media Player 11. In the **Profile** panel of the **Settings** inspector (Fig. 19.16), you can select the **Video** and **Audio** type. For the Movie Viewer example, we encoded the video using **VC-1 Streaming Broadband** and the audio using the **Default Profile**. In the **Output** inspector (Fig. 19.16), you can have Media Encoder save a thumbnail of a frame of your choosing, and have the output include one of 14 prebuilt Silverlight media player templates for your video. To start the encoding process, either select **Encode** from the **File** menu, or click the **Encode** button at the bottom of the **Media Content** inspector.

Uploading an Application to the Silverlight Streaming Service

To use the Silverlight Streaming service, you must go to silverlight.live.com and register for an account. Once you have an account, log in and select **Manage Applications** from the navigation links on the left side of the page. This page will show you what applications are currently being hosted on your account, and it also enables you to upload applications.

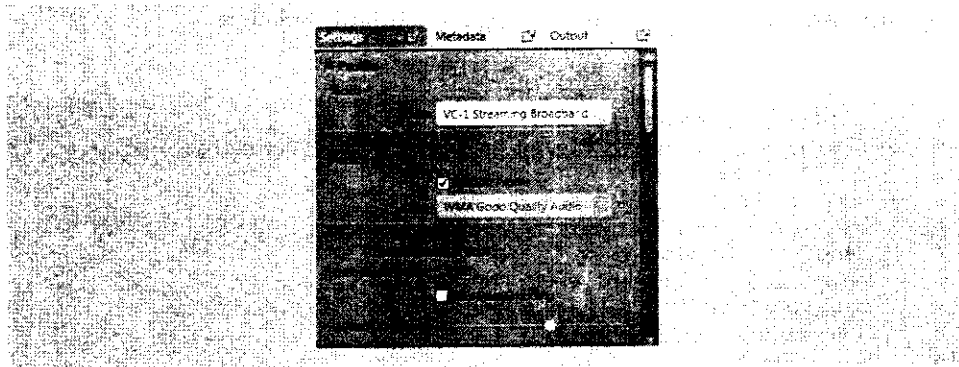


Fig. 19.16 | Microsoft's Expression Media Encoder. (Part 1 of 2.)

1. dev.live.com/terms/
2. dev.live.com/silverlight/

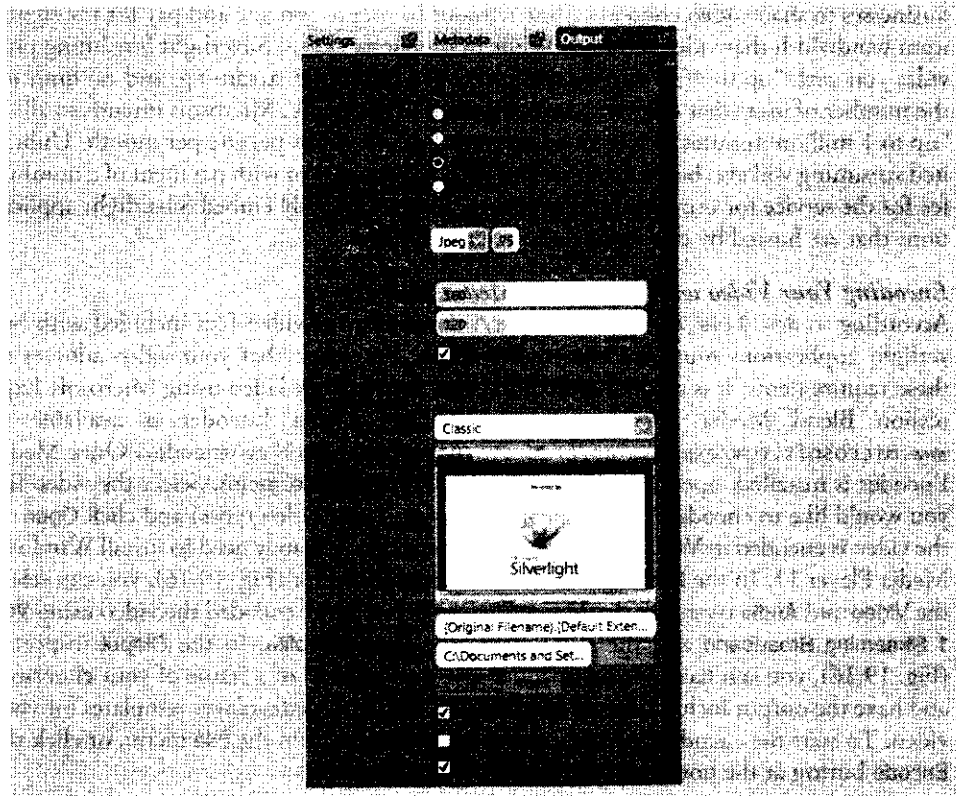


Fig. 19.16 | Microsoft's Expression Media Encoder. (Part 2 of 2.)

To upload an application, you must package it into a Zip archive. This archive must contain your XAML, your code-behind file, any media elements you use in the application, and a `manifest.xml` file (Fig. 19.17). The `manifest.xml` file specifies the filename of your XAML file (line 4), width and height (lines 5–6), and more.

```

1 <!-- Fig. 19.17: manifest.xml -->
2 <!-- Manifest for Movie Viewer on Silverlight Streaming. -->
3 <SilverlightApp>
4   <source>Page.xaml</source>
5   <width>640</width>
6   <height>480</height>
7   <inplaceInstallPrompt>true</inplaceInstallPrompt>
8   <background>#000000</background>
9   <framerate>24</framerate>
10  <version>1.0</version>
11  <iswindowless>>false</iswindowless>
12 </SilverlightApp>

```

Fig. 19.17 | Manifest for Movie Viewer on Silverlight Streaming.

On the **Manage Applications** page, click the **Upload a Silverlight Application** link. You must enter an **Application Name** and select the Zip archive you wish to upload. If the application uploads successfully after you click **Upload**, you will see the **Manage Application** page for the application. On this page, Microsoft provides instructions for adding the application to an existing web page. First, you must create a new JavaScript file to handle the `CreateSilverlight` function that adds the Silverlight application inside a `div` in your HTML. For the `div` element that you add to your HTML, make sure to set the `width` and `height` parameters to the `width` and `height` of your application. Figures 19.18 and 19.19 show the HTML and JavaScript needed to embed the Movie Viewer application that is hosted on Silverlight Streaming.

```

1 <?xml version = "1.0" encoding = "utf-8"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5 <!-- Fig. 19.18: silverlightStreaming.html -->
6 <!-- HTML wrapper for Movie Viewer hosted on Silverlight Streaming -->
7 <html xmlns = "http://www.w3.org/1999/xhtml">
8   <head>
9     <title>Movieviewer Hosted on Silverlight Streaming</title>
10    <script type = "text/javascript"
11      src = "http://agappdom.net/h/silverlight.js"></script>
12    <script type = "text/javascript"
13      src = "CreateSilverlight.js"></script>
14    <style type = "text/css">
15      silverlightHost { height: 480px;
16        width: 640px; }
17    </style>
18  </head>
19  <body>
20    <div id = "Wrapper_MovieViewer"
21      style = "width: 640px; height: 480px; overflow: hidden;"></div>
22    <script type = "text/javascript">
23      var Wrapper_MovieViewer =
24        document.getElementById( "Wrapper_MovieViewer" );
25      CreateSilverlight();
26    </script>
27  </body>
28 </html>

```

Fig. 19.18 | HTML wrapper for Movie Viewer hosted on Silverlight Streaming.

```

1 // Fig. 19.19: CreateSilverlight.js
2 // Javascript to add the Silverlight object to the Wrapper_MovieViewer div
3 function CreateSilverlight()
4 {
5   Silverlight.createObjectEx( {
6     source: "streaming://19445/MovieViewer",
7     parentElement: Wrapper_MovieViewer } );
8 }

```

Fig. 19.19 | JavaScript to add the Silverlight object to Wrapper_MovieViewer div.

19.7 Silverlight 1.1 Installation and Overview

Silverlight 1.1 uses a lightweight version of the .NET CLR (Common Language Runtime) in the browser plug-in. This allows you to program Silverlight applications in C#, Visual Basic, Python, Ruby and JavaScript. Silverlight 1.1 applications use the .NET CLR's just-in-time (JIT) compiler to compile the code to machine language, allowing for a significant improvement in performance over the interpreted JavaScript used in Silverlight 1.0 and Ajax.

To install the Silverlight 1.1 Alpha Refresh browser plug-in, go to silverlight.net/GetStarted/ and download the Silverlight 1.1 Alpha Refresh runtime for your platform. Once you have installed it, you can see some 1.1 applications in action at the website silverlight.net/themes/silverlight/community/gallerydetail.aspx?cat=2.

A chess game that serves as an excellent demonstration of the performance improvement is located at silverlight.net/samples/1.1/chess/run/default.html. This game allows you to compare the performance of a computer player coded in .NET to the performance of a computer player coded in JavaScript. As you will see, the .NET player usually wins because it can analyze many more moves than the JavaScript player in the same amount of time.

We will develop our Silverlight 1.1 applications using Microsoft Expression Blend 2 and Microsoft Visual Studio 2008. After you have installed these tools, download and install the **Silverlight Tools Alpha for Visual Studio** from go.microsoft.com/fwlink/?LinkID=89149&clid=0x409. Now, you can create a Silverlight 1.1 Alpha Refresh project.

19.8 Creating a Cover Viewer for Silverlight 1.1 Alpha

Our next example is a Deitel book cover viewer (Fig. 19.20) written in XAML (Fig. 19.21) with a Visual Basic code-behind file (Fig. 19.22) for Silverlight 1.1 Alpha

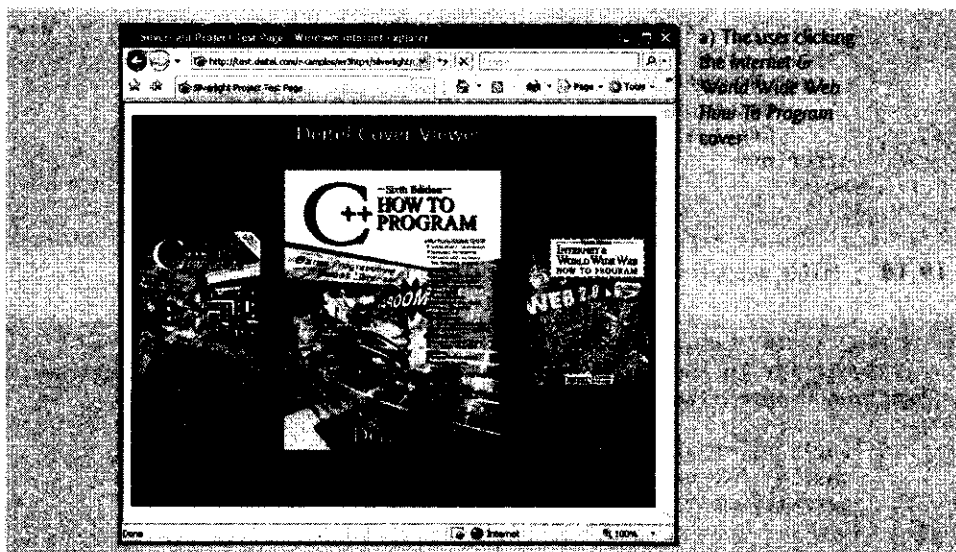


Fig. 19.20 | Deitel book-cover viewer running on Silverlight 1.1 Alpha Refresh. (Part 1 of 2.)

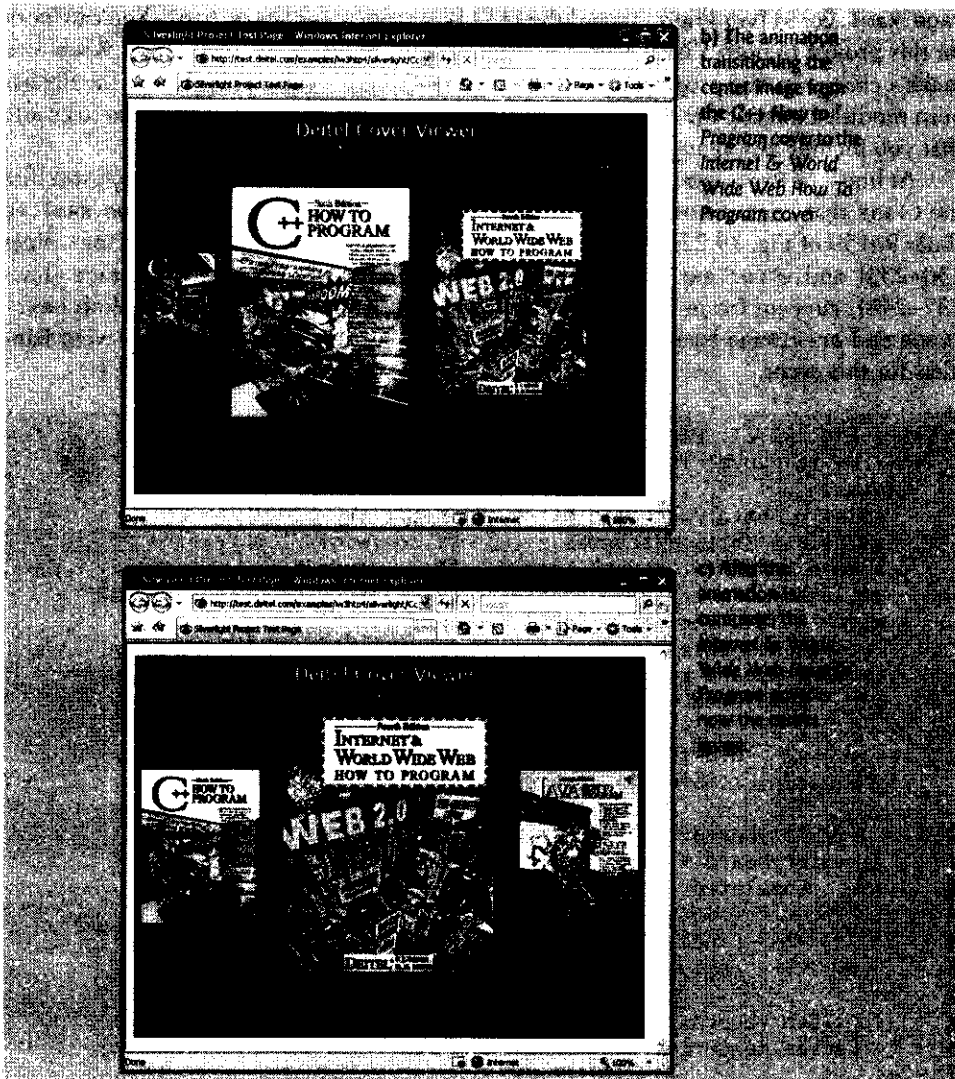


Fig. 19.20 | Deitel book-cover viewer running on Silverlight 1.1 Alpha Refresh. (Part 2 of 2.)

Refresh. This cover viewer retrieves an RSS feed containing the image URIs, and displays three covers at a time. Clicking the cover on the left or right triggers an animation that moves the cover the user clicked to the center. You can test a live version of this application at test.deitel.com/examples/iw3http4/silverlight/CoverViewer/index.html.

Creating a Silverlight 1.1 Application in Visual Studio 2008

To create a Silverlight 1.1 Alpha Refresh project, open Visual Studio 2008 and select **New Project** in the **File** menu. Next, select **Visual Basic** (for your later projects, you can also select **Visual C#**), then **Silverlight**, then specify the name and location of your project, and click **OK**. The project will initially contain a XAML file, `Page.xaml`, a code-behind file,

Page.xaml.vb, Silverlight.js and the HTML wrapper, TestPage.html. You can work on this project in both Visual Studio and Expression Blend at the same time. When you make a change in one program, then switch to the other, it will alert you that the file has been modified outside the program and prompt you to reload the file. Select **Yes** to ensure that you include any changes you made in the other program.

At line 7 of Page.xaml (Fig. 19.21), we define the **x:Class** attribute, which specifies the Class that contains our event handlers, in this case the Page class in Page.xaml.vb (lines 9–159 of Fig. 19.22). The GUI contains two TextBlocks—titleTextBlock (lines 230–233) and errorTextBlock (lines 234–236)—and three Images—prevImage (lines 237–248), currentImage (lines 249–250) and nextImage (lines 251–262). Both nextImage and prevImage have a MouseLeftButtonDown attribute, registering the event handlers for this event.

```

1  <!-- Fig. 19.21: Page.xaml -->
2  <!-- Deitel Cover Viewer in Silverlight 1.1 Alpha Refresh -->
3  <Canvas
4      xmlns="http://schemas.microsoft.com/client/2007"
5      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
6      x:Class="DeitelCoverViewer.Page" Loaded="Page_Loaded"
7      x:Class="CoverViewer.Page,Assembly=ClientBin/CoverViewer.dll"
8      Width="640" Height="480" Background="Black">
9      <Canvas.Resources>
10         <Storyboard x:Name="nextImageAnimation"
11             Completed="nextImageSwitch">
12             <!-- Lines 17-20 of the autogenerated code were -->
13             <!-- removed to save space. -->
14         </Storyboard>
15         <Storyboard
16             x:Name="prevImageAnimation"
17             Completed="prevImageSwitch">
18             <!-- Lines 101-104 of the autogenerated code were -->
19             <!-- removed to save space. -->
20         </Storyboard>
21     </Canvas.Resources>
22     <TextBlock x:Name="titleTextBlock" Canvas.Left="189.994"
23         Canvas.Top="5" FontSize="24" Width="238.308"
24         TextWrapping="Wrap" Foreground="FFFFFFFF"
25         Text="Deitel Cover Viewer" />
26     <TextBlock x:Name="errorTextBlock" Canvas.Left="0"
27         Canvas.Top="400" FontSize="12" Width="640"
28         TextWrapping="Wrap" Foreground="FFFFFFFF" />
29     <Image x:Name="prevImage" MouseLeftButtonDown="prevImageHandler"
30         Width="149.64" Height="196.652" Canvas.Left="8"
31         Canvas.Top="142" RenderTransformOrigin="0.5, 0.5">
32         <Image.RenderTransform>
33             <TransformGroup>
34                 <ScaleTransform ScaleX="1" ScaleY="1" />

```

Fig. 19.21 | Deitel Cover Viewer in Silverlight 1.1 Alpha Refresh. (Part 1 of 2.)


```

243         <SkewTransform AngleX = "0" AngleY = "0" />
244         <RotateTransform Angle = "0" />
245         <TranslateTransform X = "0" Y = "0" />
246     </TransformGroup>
247 </Image.RenderTransform>
248 </Image>
249 <Image x:Name = "currentImage" Width = "265.302" Height = "348.652"
250     Canvas.Left = "142" Canvas.Top = "43.78" />
251 <Image x:Name = "nextImage" MouseLeftButtonDown = "nextImageHandler"
252     Width = "145.64" Height = "196.652" Canvas.Left = "142"
253     Canvas.Top = "142" RenderTransformOrigin = "0.5, 0.5" />
254     <Image.RenderTransform>
255         <TransformGroup>
256             <ScaleTransform ScaleX = "1" ScaleY = "1" />
257             <SkewTransform AngleX = "0" AngleY = "0" />
258             <RotateTransform Angle = "0" />
259             <TranslateTransform X = "0" Y = "0" />
260         </TransformGroup>
261     </Image.RenderTransform>
262 </Image>
263 </Canvas>

```

Fig. 19.21 | Deitel Cover Viewer in Silverlight 1.1 Alpha Refresh. (Part 2 of 2.)

In lines 10–100 (of which lines 12–99 are not shown to save space), the `nextImageAnimation` Storyboard moves and resizes the three images so that the `nextImage` replaces the `currentImage`, the `currentImage` replaces the `prevImage`, and the `prevImage` disappears. This animation code was generated using Expression Blend. To create this animation, first click the **Create new Storyboard** button (Fig. 19.4). Name the Storyboard `nextImageAnimation`, and select the **Create as a Resource** checkbox. Then, select `nextImage` and click **Record Keyframe**. Move the time slider to 0.5 seconds, then move and resize the element so that it replaces `currentImage`. You can click the **Play** button to see a preview of the animation.

Storyboard `nextImageAnimation` has a `Completed` attribute (line 11) that specifies the event handler to be called when the animation is complete. In lines 101–227 (of which lines 103–226 are not shown to save space), the `prevImageAnimation` Storyboard moves and resizes the three images so that the `prevImage` replaces the `currentImage`, the `currentImage` replaces the `nextImage`, and the `nextImage` disappears. This animation also has a `Completed` attribute (line 102) that specifies the event handler to be called when the animation is complete.

Visual Basic Code-Behind File

The Visual Basic code-behind file, `Page.xaml.vb` (Fig. 19.22), begins by importing the class libraries the application will use (lines 3–7). Lines 9–10 specify that the `Page` class inherits methods and properties from the `Canvas` class. Lines 13–15 declare the instance variables the application will use. These include `imageURIArrayList` (a `List` of the image URIs), `currentImageIndex` (which holds the index number of the Uri in `imageURIArrayList` to be displayed as the `currentImage`), and `appRootURI` (which uses the `appRootURIGen` method (lines 151–158) to find the root URI of the application at runtime).

```

1  ' Fig. 19.22: Page.kaml.vb
2  ' VB code-behind file for Cover Viewer
3  Imports System.IO
4  Imports System.Text
5  Imports System.Windows.Browser
6  Imports System.Windows.Browser.Net
7  Imports System.Collections.Generic
8
9  Public Class Page
10     Inherits Canvas
11
12     ' Instance variables
13     Dim currentIndex = 0 ' Initialize index of currentImage as 0
14     Dim imageURIArrayList As New List(Of Uri)() ' Create ArrayList of URIs
15     Dim appRootURI = appRootURIGen() ' Store application root URI
16
17     Public Sub Page_Loaded(ByVal o As Object, ByVal e As EventArgs)
18         ' Required to initialize variables
19         InitializeComponent()
20
21         Dim httpRequest As New _
22             BrowserHttpWebRequest( _
23                 New Uri(appRootURI + "bookCoversRSS.xml"))
24
25         ' Save response in variable
26         Dim httpResponse = httpRequest.GetResponse()
27
28         ' Save response stream in variable
29         Dim httpResponseStream = httpResponse.GetResponseStream()
30
31         Dim currentImageURI ' Store image URI
32
33         ' Create an XmlReader to parse the response stream
34         Using xmlReader As XmlReader = xmlReader.Create( _
35             New StreamReader(httpResponseStream))
36
37             ' Start reading response stream, exit loop when done
38             While (xmlReader.Read())
39
40                 ' Find item element in response stream
41                 If ((xmlReader.IsStartElement()) And _
42                     ("item" = xmlReader.LocalName)) Then
43
44                     ' Create an XmlReader for item element
45                     Using itemXMLReader As XmlReader = _
46                         xmlReader.ReadSubtree()
47
48                         ' Start reading item element, exit loop when done
49                         While (itemXMLReader.Read())
50
51                             ' Find image child element of item
52                             If (itemXMLReader.IsStartElement()) Then
53                                 If ("image" = itemXMLReader.LocalName) Then

```

Fig. 19.22 | VB code-behind file for Cover Viewer. (Part I of 3.)

```

54
55         ' Save Uri of image into ArrayList
56         currentImageURI = appRootURI + _
57         itemXMLReader.ReadElementContentAsString
58         imageURIArrayList.Add( _
59             New Uri(currentImageURI))
60     End If
61 End If
62 End While
63 End Using
64 End If
65 End While
66 End Using
67
68 ' Close BrowserHttpRequest
69 httpResponse.Close()
70
71 ' Initialize currentImage and nextImage Sources
72 currentImage.Source = imageURIArrayList(currentImageIndex)
73 nextImage.Source = imageURIArrayList(currentImageIndex + 1)
74
75 Catch ex As Exception
76     errorTextBlock.Text = "Error: " & ex.Message
77 End Try
78 End Sub ' Page_Loaded
79
80 ' Handle nextImageAnimation's Completed event
81 Private Sub nextImageSwitch(ByVal sender As Object, _
82     ByVal e As EventArgs)
83     nextImageAnimation.Stop()
84
85     ' Test if at end of images
86     If (currentImageIndex = (imageURIArrayList.Count - 2)) Then
87         currentImageIndex += 1 ' Increment currentImageIndex
88
89     ' Set Source of prevImage and currentImage
90     prevImage.Source = imageURIArrayList(currentImageIndex - 1)
91     currentImage.Source = imageURIArrayList(currentImageIndex)
92     nextImage.Opacity = 0 ' Hide nextImage
93 Else
94     currentImageIndex += 1 ' Increment currentImageIndex
95
96     ' Set Source of prevImage, currentImage and nextImage
97     prevImage.Source = imageURIArrayList(currentImageIndex - 1)
98     currentImage.Source = imageURIArrayList(currentImageIndex)
99     nextImage.Source = imageURIArrayList(currentImageIndex + 1)
100    prevImage.Opacity = 1 ' Show prevImage
101 End If
102 End Sub ' nextImageSwitch
103
104 ' Handle prevImageAnimation's Completed event
105 Private Sub prevImageSwitch(ByVal sender As Object, _
106     ByVal e As EventArgs)

```

Fig. 19.22 | VB code-behind file for Cover Viewer. (Part 2 of 3.)

```

107     prevImageAnimation.Stop()
108
109     ' Test IF at beginning of images
110     IF (currentImageIndex = 1) Then
111         currentImageIndex -= 1 ' Decrement currentImageIndex
112         prevImage.Opacity = 0 ' Hide prevImage
113
114         ' Set source of currentImage and nextImage
115         currentImage.Source = ImageURLArrayList(currentImageIndex)
116         nextImage.Source = ImageURLArrayList(currentImageIndex + 1)
117     Else
118         currentImageIndex -= 1 ' Decrement currentImageIndex
119
120         ' Set source of prevImage, currentImage and nextImage
121         prevImage.Source = ImageURLArrayList(currentImageIndex - 1)
122         currentImage.Source = ImageURLArrayList(currentImageIndex)
123         nextImage.Source = ImageURLArrayList(currentImageIndex + 1)
124         nextImage.Opacity = 1 ' Show nextImage
125     End IF
126 End Sub ' prevImageSwitch
127
128 ' Handle nextImage's MouseLeftButtonDown event
129 Private Sub nextImageHandler(ByVal sender As Object,
130     ByVal e As EventArgs)
131
132     ' Make sure there are more images to the right
133     IF (currentImageIndex < (ImageURLArrayList.Count - 1)) Then
134         nextImageAnimation.Begin()
135     End IF
136 End Sub ' nextImageHandler
137
138 ' Handle prevImage's MouseLeftButtonDown event
139 Private Sub prevImageHandler(ByVal sender As Object,
140     ByVal e As EventArgs)
141
142     ' Make sure there are more images to the left
143     IF (currentImageIndex > 1) Then
144         prevImageAnimation.Begin()
145     ElseIf (currentImageIndex > 0) Then
146         prevImageAnimation.Begin()
147     End IF
148 End Sub ' prevImageHandler
149
150 ' Generate root URI of application
151 Private Function appRootURIGen() As String
152
153     ' Find root directory of application
154     Dim path = HtmlPage.DocumentUri.AbsolutePath
155     Dim lastSlash = path.LastIndexOf("/")
156     path = path.Substring(0, lastSlash + 1)
157     Return "http://" & HtmlPage.DocumentUri.Host & path
158 End Function ' appRootURIGen
159 End Class ' Page

```

Fig. 19.22 | VB code-behind file for Cover Viewer. (Part 3 of 3.)

In method `Page_Loaded` (lines 17–78), line 19 initializes the application using the `InitializeComponent` method located in the autogenerated `Page.g.vb` file (located in the `obj\Debug` directory). This file takes any XAML elements that have an `x:Name` attribute, and uses the `FindName` method to map each element to a variable of the same name. This means that we do not have to do this manually, as we did for the Silverlight 1.0 **Movie Viewer** example. It also allows us to use Visual Studio's IntelliSense feature to autocomplete XAML element names in our code-behind file.

Lines 20–77 try to download an RSS file, `bookCoversRSS.xml` (Fig. 19.23), and create an array of image URIs. First, lines 21–23 create a `BrowserHttpRequest` object that downloads the RSS file located at the URI created by concatenating the `appRootURI` variable with `bookCoversRSS.xml`. Note that the `BrowserHttpRequest` object does not currently support cross-domain requests, so the application and the RSS file must be located on the same server. Lines 26 and 29 get the object used to manipulate the request's response, then get the stream associated with that object. Lines 34–35 create an `XmlReader` object to parse the RSS content. The `XmlReader` class provides read-only access to the elements in an XML document. Lines 38–65 contain a `while` loop in which the condition remains `True` until the `XmlReader` has reached the end of the RSS. Lines 41–42 search for an `item` element in the RSS, and lines 45–53 read the contents of that element and search for an `image` element inside the `item` element. Upon finding an `image` element, lines 56–59 add the contents of the `image` element (the image's filename) to the `imageURIArrayList` as a complete URI including the application's root URI (`appRootURI`). Line 69 closes the `BrowserHttpRequest`. Lines 72–73 set the `Source` attribute of the `currentImage` and `nextImage` to the first and second elements of the `imageURIArrayList`. Lines 75–76 catch any exceptions and display the error message in the `errorTextBlock`.

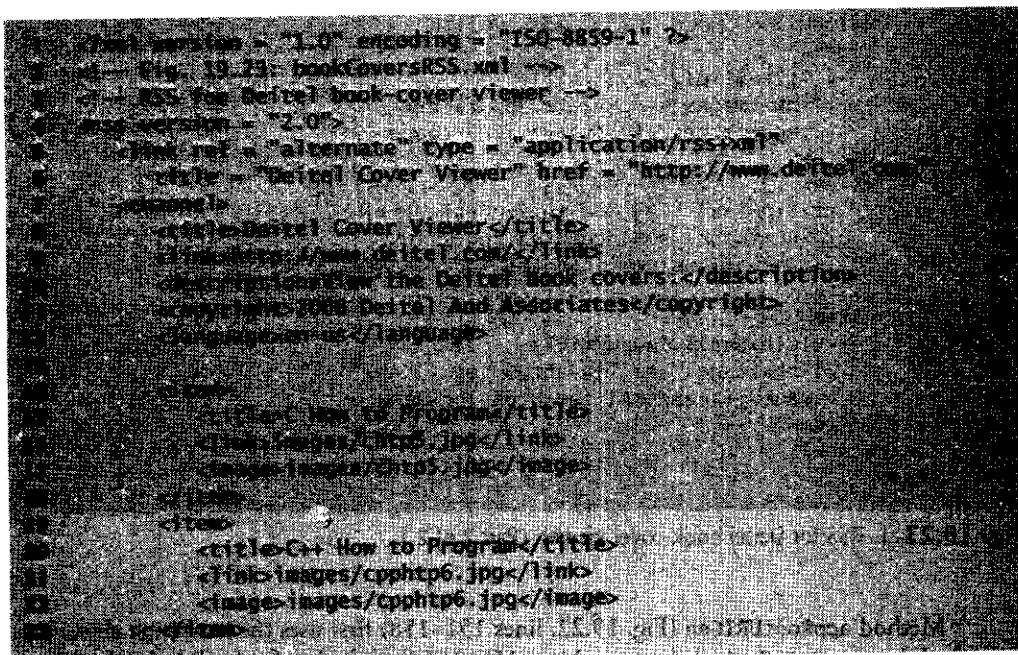


Fig. 19.23 | RSS for Deitel book-cover viewer. (Part 1 of 2.)

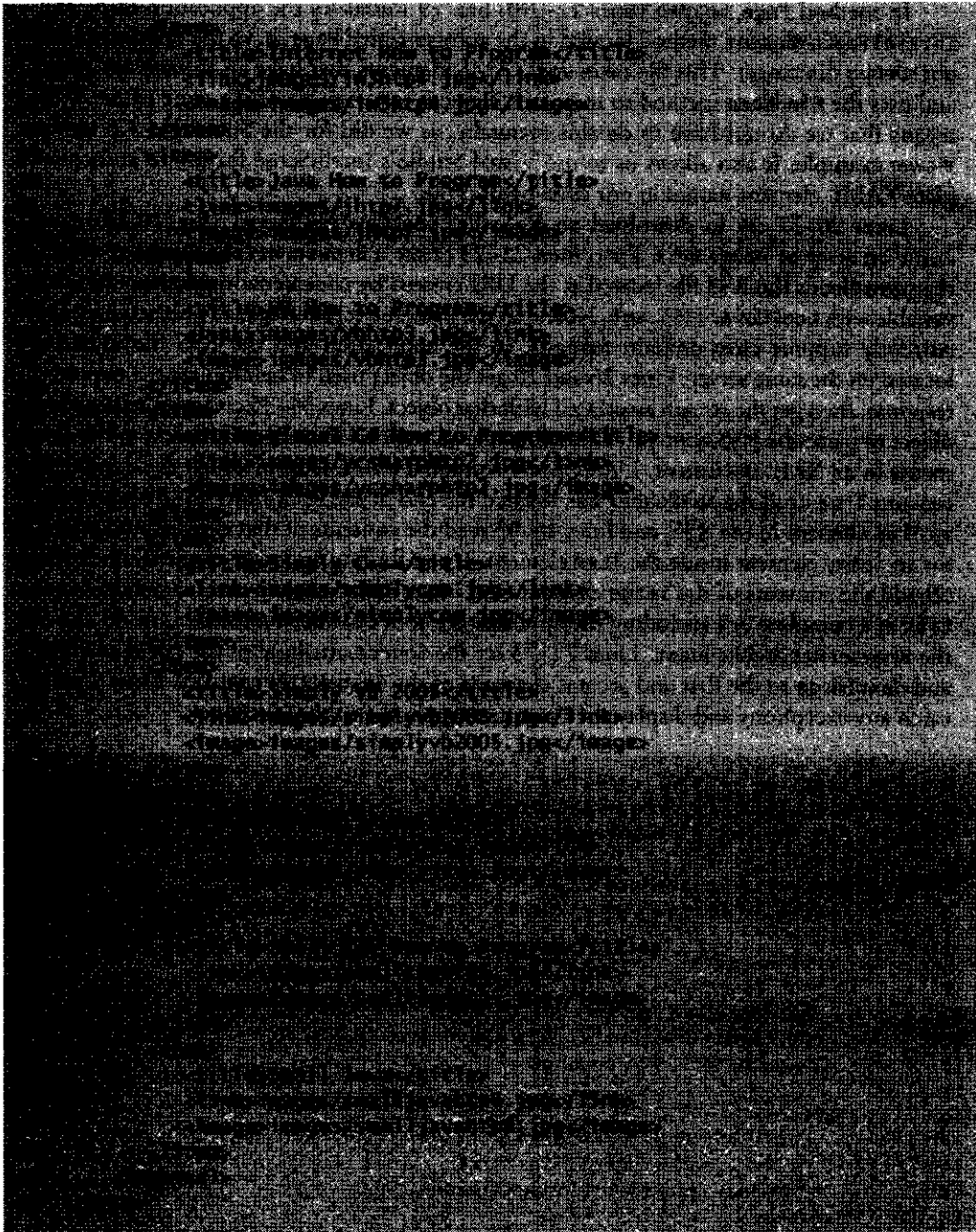


Fig. 19.23 | RSS for Deitel book-cover viewer. (Part 2 of 2.)

Method `appRootURIGen` (Fig. 19.22, lines 151–158) first uses the `HtmlPage` element to find the `AbsolutePath` of the page. Lines 155–156 find the last forward slash (/) of the Uri and save the Uri up to that last slash as a string, using the `Substring` method. Line

157 returns a string concatenating "http://", the Silverlight application's Host (the domain name or IP address of the server) and the path string.

Method `nextImageHandler` (lines 129–136) handles `nextImage`'s `MouseDown` event. Line 133 checks whether there are, in fact, additional book covers to the right. If so, line 134 begins the `nextImageAnimation` Storyboard. Upon completion, this Storyboard will call the `nextImageSwitch` method (lines 81–102). Line 86 checks whether there is only one more book cover to the right, in which case it will increment the `currentIndex` by one (line 87), update the `Source` of `prevImage` and `currentImage` (90–91), and hide `nextImage` (line 92). If there is more than one book cover to the right, lines 94–100 will increment the `currentIndex` by one (line 94), update the `Source` of all three Images (lines 97–99), and ensure that `prevImage` is visible (line 100), in case the user is going from the first book cover (where `prevImage` would be hidden) to the second book cover. Methods `prevImageHandler` (lines 139–148) and `prevImageSwitch` (lines 105–126) provide the corresponding functionality for `prevImage`.

19.9 Building an Application with Third-Party Controls

Though Silverlight 1.1 Alpha Refresh does not yet include pre-built controls, a number of third-party control libraries have been created. One such third-party library is Netika's GOA WinForms library for Silverlight. This library is an implementation of .NET's `System.Windows.Forms` library for both Silverlight and Flash. This allows us to create Silverlight applications by using .NET desktop applications as templates. The free version of GOA WinForms includes 40+ controls, including buttons, text boxes, calendars and more. Netika's website at www.netikatech.com includes demos and documentation for all the controls. To download the library, go to www.netikatech.com/downloads and select the standard Silverlight version of GOA WinForms. After installation, open Visual Studio 2008 and create a new project. Select **Visual Basic**, then **GOA WinForms VB Application in My Templates**. Name this project `InterestRateCalculator`, as we will be creating a Silverlight application that calculates interest. For a GOA WinForms project, the Visual Basic code-behind file is located in `MyForm.vb`. In this file, you will find an `InitializeComponent` function (lines 27–42) that creates a `Button`. Select **Build InterestRateCalculator** from the **Build** menu, then open `TestPage.html` in your browser to see a sample button.

Open up the `InterestRateCalculatorForWindows` project from the examples directory. We are going to be creating a Silverlight application from this desktop application (Fig. 19.24). First, build and run the project to see how the application looks on the desktop. Next, replace the `InitializeComponent` function in the `InterestRateCalculator` project's `MyForm.vb` with the `InitializeComponent` in the `InterestRateCalculatorForWindows` project's `InterestRateCalculatorForWindows.Designer.vb`. Then replace the `Friend WithEvents` line (line 26) in the `InterestRateCalculator` project's `MyForm.vb` with the `Friend WithEvents` lines (lines 139–147) in the `InterestRateCalculatorForWindows` project's `InterestRateCalculatorForWindows.Designer.vb` file. Finally, copy the `btnCalculate_Click` function from the `InterestRateCalculatorForWindows` project's `InterestRateCalculatorForWindows.vb` into the `MyForm` class in the `InterestRateCalculator` project's `MyForm.vb`.

Try to build the `InterestRateCalculator` project. You will see several errors. This is because not every property of the Windows Form controls has been implemented in GOA WinForms. Looking at Fig. 19.25, you will see that we commented out lines 47, 60, 82,

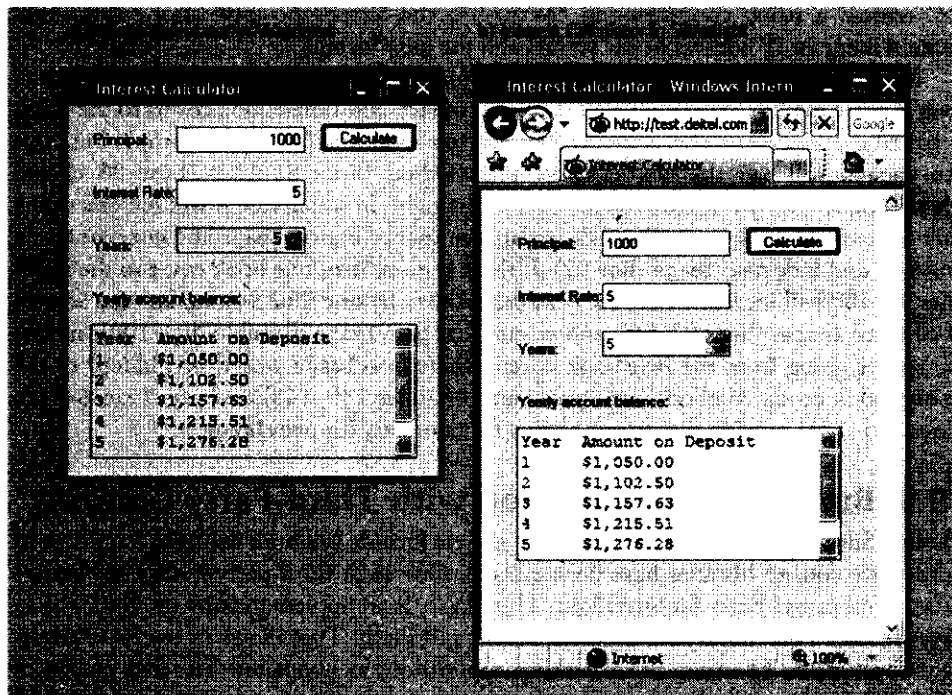


Fig. 19.24 | Interest Calculator in Windows and Silverlight

100, 118 and 128–129. These lines all accessed properties not supported in GOA WinForms. We kept these lines as comments to show you the relatively easy process of converting a Visual Basic desktop application to a Silverlight 1.1 application when using GOA WinForms controls.

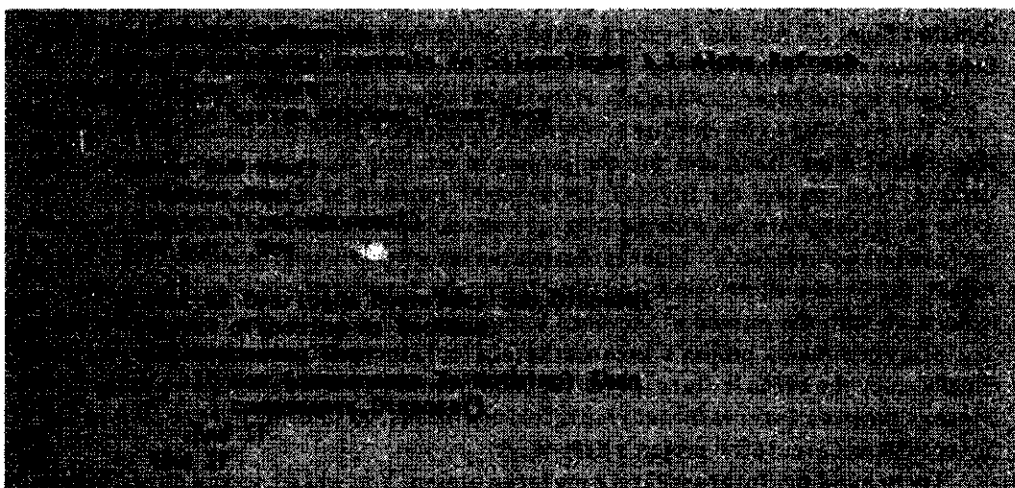


Fig. 19.25 | Using third-party controls in Silverlight 1.1 Alpha Refresh. (Part 1 of 5.)

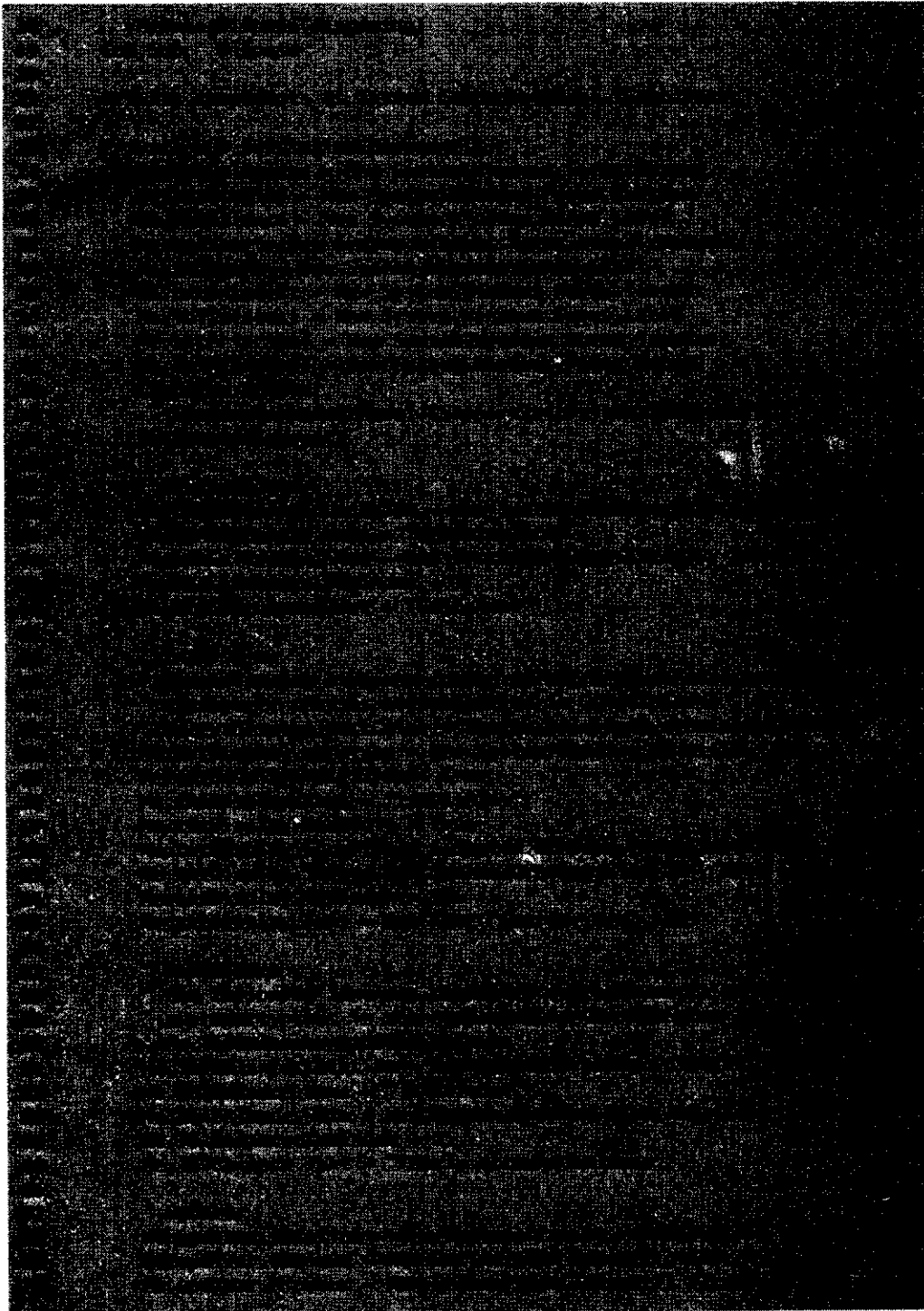


Fig. 19.25 | Using third-party controls in Silverlight 1.1 Alpha Refresh. (Part 2 of 5.)

```

Me.updYear.Name = "updYear"
Me.updYear.ReadOnly = True
Me.updYear.Size = New System.Drawing.Size(100, 20)
Me.updYear.TabIndex = 20
Me.updYear.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Center
Me.updYear.Value = New Decimal(100)

' lblYears
' the following line was commented out because it was
' a property that is not supported in Silverlight
Me.lblYears.AutoSize = True
Me.lblYears.Location = New System.Drawing.Point(100, 100)
Me.lblYears.Name = "lblYears"
Me.lblYears.Size = New System.Drawing.Size(100, 20)
Me.lblYears.TabIndex = 13
Me.lblYears.Text = "Years:"

' txtInterest
Me.txtInterest.Location = New System.Drawing.Point(100, 150)
Me.txtInterest.Name = "txtInterest"
Me.txtInterest.Size = New System.Drawing.Size(100, 20)
Me.txtInterest.TabIndex = 12
Me.txtInterest.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Center

' lblInterest
' the following line was commented out because it was
' a property that is not supported in Silverlight
Me.lblInterest.AutoSize = True
Me.lblInterest.Location = New System.Drawing.Point(100, 180)
Me.lblInterest.Name = "lblInterest"
Me.lblInterest.Size = New System.Drawing.Size(100, 20)
Me.lblInterest.TabIndex = 11
Me.lblInterest.Text = "Interest:"

' txtPrincipal
Me.txtPrincipal.Location = New System.Drawing.Point(100, 230)
Me.txtPrincipal.Name = "txtPrincipal"
Me.txtPrincipal.Size = New System.Drawing.Size(100, 20)
Me.txtPrincipal.TabIndex = 10
Me.txtPrincipal.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Center

' lblPrincipal
' the following line was commented out because it was
' a property that is not supported in Silverlight
Me.lblPrincipal.AutoSize = True
Me.lblPrincipal.Location = New System.Drawing.Point(100, 260)
Me.lblPrincipal.Name = "lblPrincipal"
Me.lblPrincipal.Size = New System.Drawing.Size(100, 20)
Me.lblPrincipal.TabIndex = 9
Me.lblPrincipal.Text = "Principal:"
    
```

Fig. 19.25 | Using third-party controls in Silverlight 1.1 Alpha Refresh. (Part 3 of 5.)

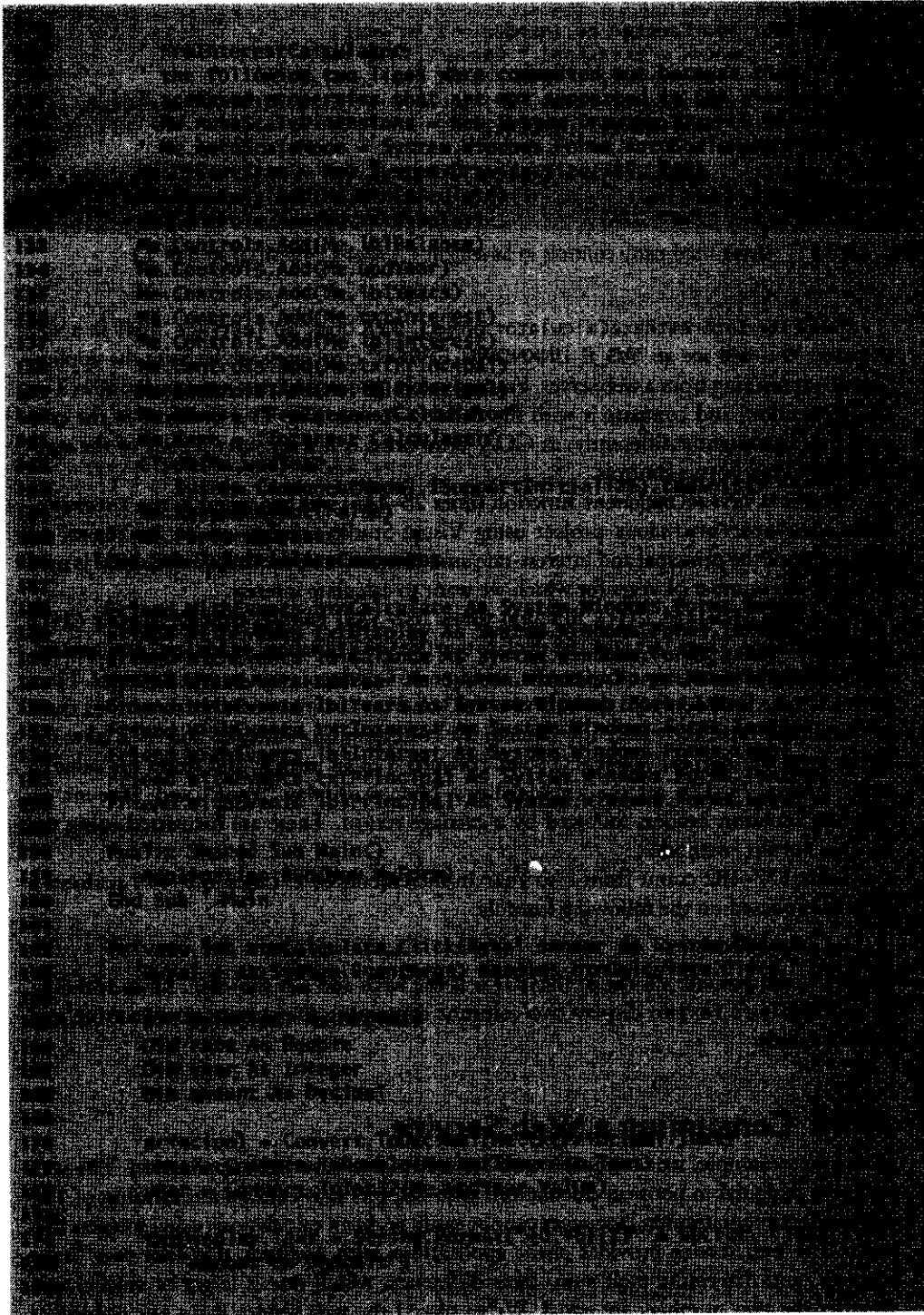


Fig. 19.25 | Using third-party controls in Silverlight 1.1 Alpha Refresh. (Part 4 of 5.)

```

For yearCounter As Integer = 1 To year
    amount = principal * Convert.ToDecimal( _
        Math.Pow(1 + rate / 100, yearCounter))
    txtDisplay.Text &= String.Format("{0,-6:D}{1:C}" & vbCrLf, _
        yearCounter, amount)
Next
Sub btnCalculate_Click
    ' MyForm

```

Fig. 19.25 | Using third-party controls in Silverlight 1.1 Alpha Refresh. (Part 5 of 5.)

Build the `InterestRateCalculator` project, then open up `TestPage.html` in a web browser. You will see an `"AG_E_UNKNOWN_ERROR"` error message because the application is not running from a web server. You can safely ignore this error message for now. Test the application, and compare it with the desktop version (Fig. 19.24). Some of the controls function slightly differently, as `GOA WinForms` is not an exact replica of the standard Windows Forms controls.

The `InitializeComponent` function (lines 23–146) was generated in the `InterestRateCalculatorForWindows` project using Visual Studio's design mode. `TextBoxes` are used to input the principal and interest-rate amounts, and a `NumericUpDown` control is used to input the number of years for which we want to calculate interest.

The `btnCalculate_Click` function (lines 162–183) handles `btnCalculate`'s `Click` event (line 163). Lines 165 and 168 declare two `Decimal` variables, `principal` and `amount`. Line 166 declares `rate` as type `Double`, and line 167 declares `year` as type `Integer`. Lines 170–171 take the `Text` from the `txtPrincipal` and `txtInterest` text boxes, convert them to the correct type, then store the value in the corresponding variable. Line 172 takes the `Value` from the `updYear` `NumericUpDown`, converts it to an integer, and stores the value to `year`. Lines 174–175 set the `txtDisplay`'s `Text` to display "Year" and "Amount on Deposit" column headers followed by a carriage return. These are formatted using the `String.Format` method.

Lines 177–182 count from 1 to `year` in increments of 1. Lines 178–179 perform a calculation based on the following formula:

$$a = p(1 + r)^n$$

where a is the amount, p is the principal, r is the rate and n is the year. Lines 180–181 set `txtDisplay`'s `Text` to display two columns containing the current `yearCounter` and `amount` values.

19.10 Consuming a Web Service

In the next example, we consume a web service from a Silverlight application. The web service is designed to perform calculations with integers that contain a maximum of 100 digits. Most programming languages cannot easily perform calculations using integers this large. The web service provides client applications with methods that take two "huge integers" and determine their sum, their difference, which one is larger or smaller and whether the two numbers are equal. We've placed the web service is on our website at `test.deitel.com/hugeinteger/hugeinteger.asmx`.

We provide a Visual Basic program that consumes this web service. We will create a Silverlight application using that application's code, then we'll add a proxy class to the project that allows the Silverlight application to access the web service. The proxy class (or proxy) is generated from the web service's WSDL file and enables the client to call web methods over the Internet. The proxy class handles all the details of communicating with the web service. The proxy class is hidden from you by default—you can view it in the **Solution Explorer** by clicking the **Show All Files** button. The proxy class's purpose is to make clients think that they are calling the web methods directly.

When you add a web reference to the Silverlight project, Visual Studio will generate the appropriate proxy class. You will then create an instance of the proxy class and use it to call the web service's methods. First, create a new **GOA WinForms VB Application** named **HugeInteger** in Visual Studio 2008, then perform the following steps:

Step 1: Opening the Add Web Reference Dialog

Right click the project name in the **Solution Explorer** and select **Add Web Reference...** (Fig. 19.26).

Step 2: Locating Web Services on Your Computer

In the **Add Web Reference** dialog that appears (Fig. 19.26), enter `http://test.deitel.com/hugeinteger/hugeinteger.asmx` in the URL field and press **Go**. You will see a list of the operations that the **HugeInteger** web service provides. Note that for the application to work, it must reside on the same server as the web service, because Silverlight 1.1 does not yet allow for cross-domain requests. These steps demonstrate the process we went through to create the application on our server at `test.deitel.com/examples/iw3http4/silverlight/HugeInteger/`.

Step 3: Adding the Web Reference

Add the web reference by clicking the **Add Reference** button (Fig. 19.27).

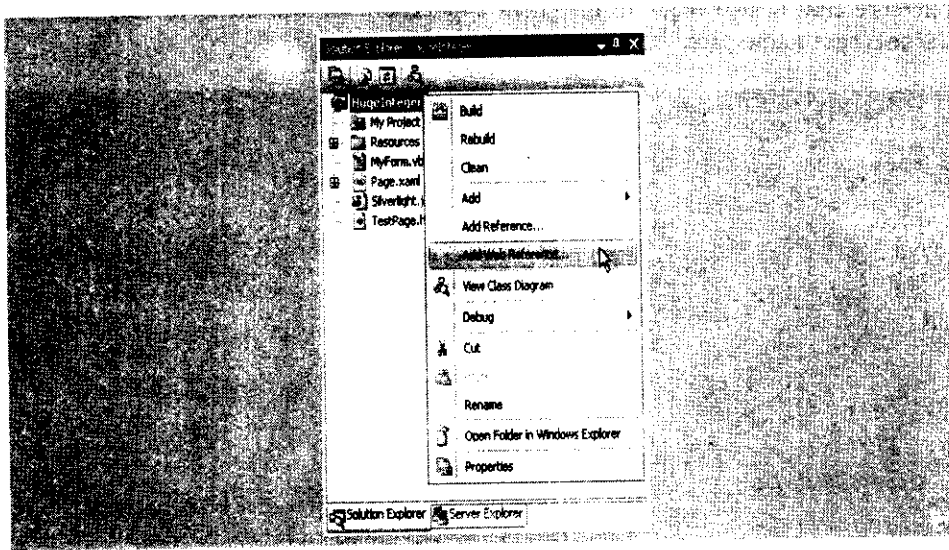


Fig. 19.26 | Adding a web service reference to a project.

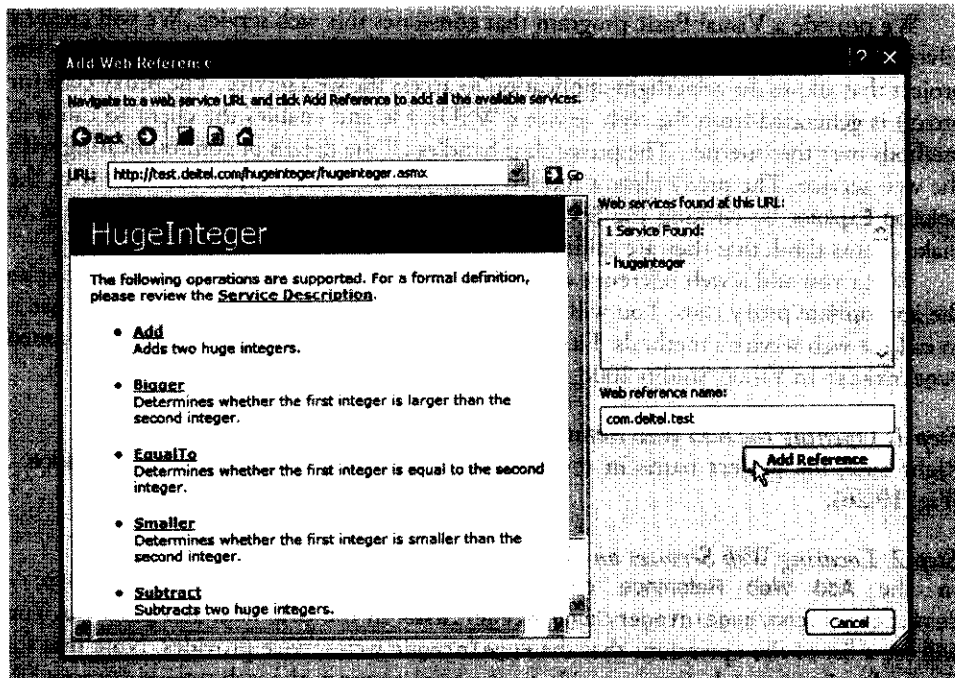


Fig. 19.27 | Web reference selection and description.

Step 4: Viewing the Web Reference in the Solution Explorer

The **Solution Explorer** (Fig. 19.28) should now contain a **Web References** folder with a node named after the domain name where the web service is located. In this case, the name is `com.deitel.test` because we are using a web service from `test.deitel.com`. When we reference class `HugeInteger` in the client application, we will do so through the `com.deitel.test` namespace.

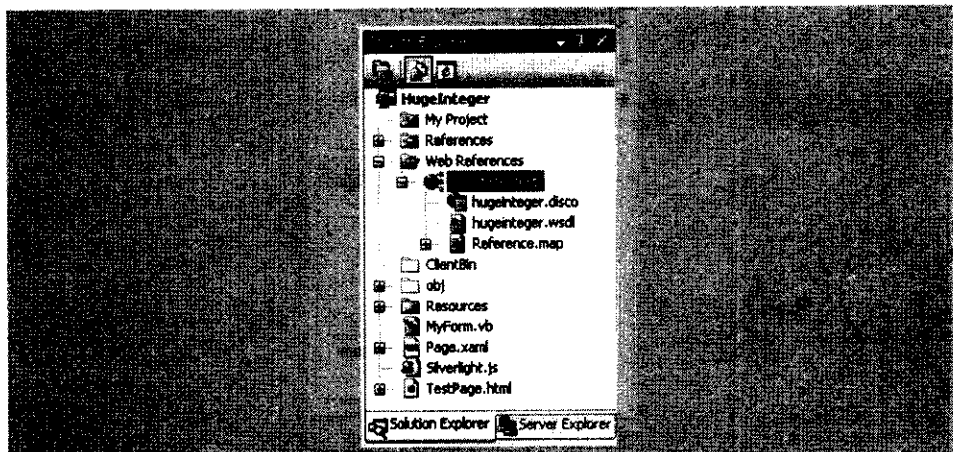


Fig. 19.28 | Solution Explorer after adding a web reference to a project.

19.10.1 Consuming the HugeInteger Web Service

Now, copy the `InitializeComponent` function (lines 14–109) and `Private WithEvents` section (lines 111–119) from `HugeIntegerForWindows.Designer.vb` in the `HugeIntegerForWindows` project to `MyForm.vb` in the `HugeInteger` project. Delete the old `InitializeComponent` function and `Friend WithEvents` line in `MyForm.vb`. Then, copy lines 6–117 from `HugeIntegerForWindows.vb` in the `HugeIntegerForWindows` project into the `MyForm Class` located in `MyForm.vb` in the `HugeInteger` project. If you try to build the project now, you will notice that the code is trying to access properties not supported by GOA WinForms. In Fig. 19.29, we commented out lines 38–41, 85–86, 95–95, 105–107 and 118–119 to remove statements that access unsupported properties in GOA WinForms. If you were running this application from the same server as the web service, you could now build this application and run it by opening `TestPage.html`. Try the completed application at test.deitel.com/examples/iw3http4/silverlight/HugeInteger/.

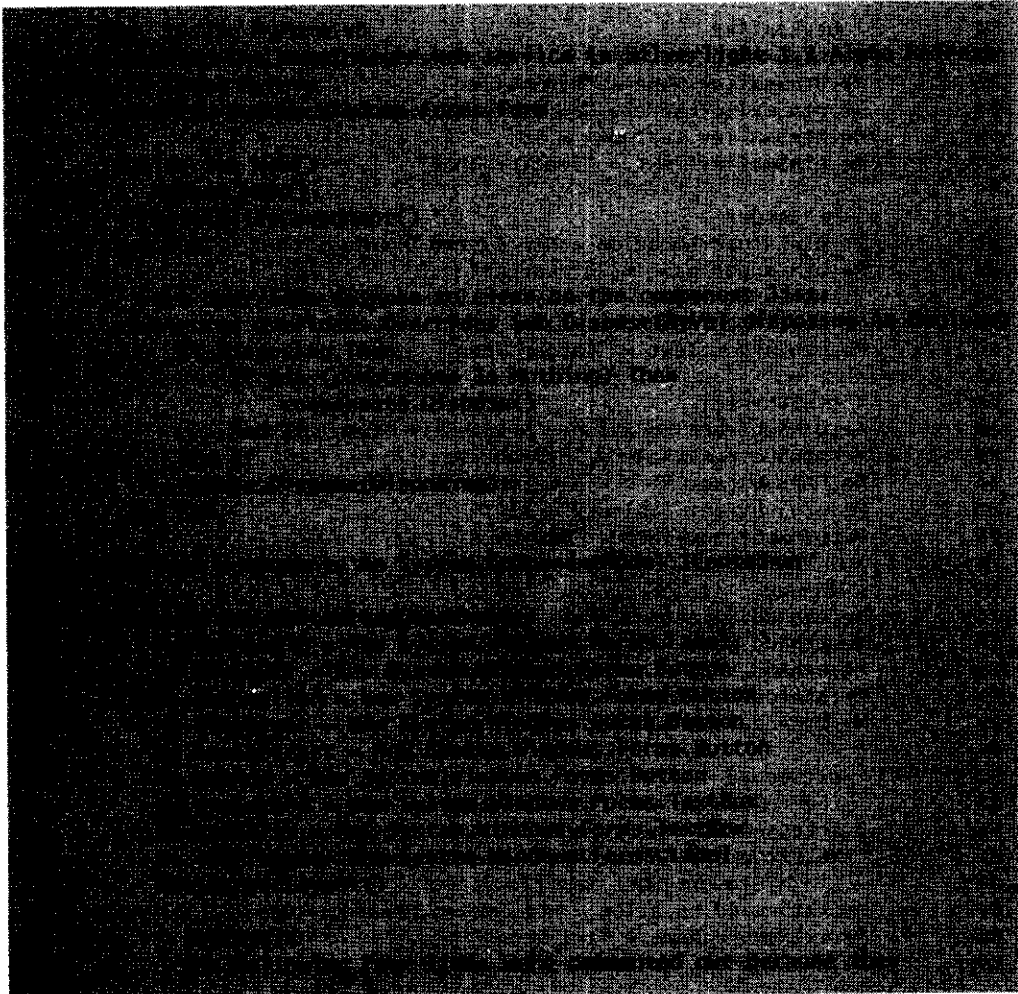


Fig. 19.29 | Consuming the HugeInteger web service in Silverlight 1.1 Alpha. (Part 1 of 6.)

```

37     * accessed properties that are not supported in COA WinForms
38     * Me.lblResult.BorderStyle = System.Windows.Forms.BorderStyle
39     * FixedSingle()
40     * Me.lblResult.Font = New System.Drawing.Font(
41     * "Microsoft Sans Serif", 9.75!)
42     Me.lblResult.Location = New System.Drawing.Point(13, 122)
43     Me.lblResult.Name = "lblResult"
44     Me.lblResult.Size = New System.Drawing.Size(714, 37)
45     Me.lblResult.TabIndex = 17
46
47     * btnEqual
48     Me.btnEqual.Location = New System.Drawing.Point(562, 91)
49     Me.btnEqual.Name = "btnEqual"
50     Me.btnEqual.Size = New System.Drawing.Size(85, 23)
51     Me.btnEqual.TabIndex = 16
52     Me.btnEqual.Text = "Equal"
53
54     * btnSmaller
55     Me.btnSmaller.Location = New System.Drawing.Point(445, 91)
56     Me.btnSmaller.Name = "btnSmaller"
57     Me.btnSmaller.Size = New System.Drawing.Size(85, 23)
58     Me.btnSmaller.TabIndex = 15
59     Me.btnSmaller.Text = "Smaller Than"
60
61     * btnLarger
62     Me.btnLarger.Location = New System.Drawing.Point(328, 91)
63     Me.btnLarger.Name = "btnLarger"
64     Me.btnLarger.Size = New System.Drawing.Size(85, 23)
65     Me.btnLarger.TabIndex = 14
66     Me.btnLarger.Text = "Larger Than"
67
68     * btnSubtract
69     Me.btnSubtract.Location = New System.Drawing.Point(211, 91)
70     Me.btnSubtract.Name = "btnSubtract"
71     Me.btnSubtract.Size = New System.Drawing.Size(85, 23)
72     Me.btnSubtract.TabIndex = 13
73     Me.btnSubtract.Text = "Subtract"
74
75     * btnAdd
76     Me.btnAdd.Location = New System.Drawing.Point(94, 91)
77     Me.btnAdd.Name = "btnAdd"
78     Me.btnAdd.Size = New System.Drawing.Size(85, 23)
79     Me.btnAdd.TabIndex = 12
80     Me.btnAdd.Text = "Add"
81
82     * txtSecond
83     * the following two lines were commented out because they
84     * accessed a property that is not supported in COA WinForms
85     * Me.txtSecond.Font = New System.Drawing.Font(
86     * "Microsoft Sans Serif", 9.75!)
87     Me.txtSecond.Location = New System.Drawing.Point(13, 63)
88     Me.txtSecond.Name = "txtSecond"
89     Me.txtSecond.Size = New System.Drawing.Size(714, 22)

```

Fig. 19.29 | Consuming the HugeInteger web service in Silverlight 1.1 Alpha. (Part 2 of 6.)


```

90 Me.txtSecond.TabIndex = 11
91
92 * txtFirst
93 * the following two lines were commented out because they
94 * accessed a property that is not supported in COA WinForms
95 * Me.txtFirst.Font = New System.Drawing.Font(
96 *     "Microsoft Sans Serif", 9.75!)
97 Me.txtFirst.Location = New System.Drawing.Point(13, 35)
98 Me.txtFirst.Name = "txtFirst"
99 Me.txtFirst.Size = New System.Drawing.Size(714, 22)
100 Me.txtFirst.TabIndex = 10
101
102 * lblPrompt
103 * the following three lines were commented out because they
104 * accessed properties that are not supported in COA WinForms
105 * Me.lblPrompt.AutoSize = True
106 * Me.lblPrompt.Font = New System.Drawing.Font(
107 *     "Microsoft Sans Serif", 9.75!)
108 Me.lblPrompt.Location = New System.Drawing.Point(13, 16)
109 Me.lblPrompt.Name = "lblPrompt"
110 Me.lblPrompt.Size = New System.Drawing.Size(339, 16)
111 Me.lblPrompt.TabIndex = 9
112 Me.lblPrompt.Text = "Please enter two positive numbers" &
113     " up to 100 digits each."
114
115 * UsingHugeIntegerWebService
116 * the following two lines were commented out because they
117 * accessed properties that are not supported in COA WinForms
118 * Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
119 * Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
120 Me.ClientSize = New System.Drawing.Size(740, 175)
121 Me.Controls.Add(Me.lblResult)
122 Me.Controls.Add(Me.btnEqual)
123 Me.Controls.Add(Me.btnSmaller)
124 Me.Controls.Add(Me.btnLarger)
125 Me.Controls.Add(Me.btnSubtract)
126 Me.Controls.Add(Me.btnAdd)
127 Me.Controls.Add(Me.txtSecond)
128 Me.Controls.Add(Me.txtFirst)
129 Me.Controls.Add(Me.lblPrompt)
130 Me.Name = "UsingHugeIntegerWebService"
131 Me.Text = "Using Huge Integer Web Service"
132 Me.ResizeLayout(False)
133 Me.PerformLayout()
134
135 End Sub
136
137 Private WithEvents lblResult As System.Windows.Forms.Label
138 Private WithEvents btnEqual As System.Windows.Forms.Button
139 Private WithEvents btnSmaller As System.Windows.Forms.Button
140 Private WithEvents btnLarger As System.Windows.Forms.Button
141 Private WithEvents btnSubtract As System.Windows.Forms.Button
142 Private WithEvents btnAdd As System.Windows.Forms.Button
143 Private WithEvents txtSecond As System.Windows.Forms.TextBox

```

Fig. 19.29 | Consuming the HugeInteger web service in Silverlight 1.1 Alpha. (Part 3 of 6.)

```

197 Private WithEvents txtFirst As System.Windows.Forms.TextBox
198 Private WithEvents txtSecond As System.Windows.Forms.TextBox
199
200 Private Shared Sub Main()
201     Application.Run(New Form1)
202 End Sub
203
204 ' declares a reference to Web service
205 Private remoteInteger As New com.deitel.test.HugeInteger
206
207 ' character to trim from strings
208 Private zeros As Char() = New Char() {"0"}
209
210 ' adds two numbers input by user
211 Private Sub btnAdd_Click(ByVal sender As System.Object,
212     ByVal e As System.EventArgs) Handles btnAdd.Click
213     ' make sure numbers do not exceed 100 digits and are at
214     ' least 100 digits long, which would cause an overflow
215     If (txtFirst.Text.Length < 100 Or txtSecond.Text.Length
216         Or (txtFirst.Text.Length = 100 And
217             txtSecond.Text.Length = 100) Then
218         MessageBox.Show(
219             "Huge integers must not be more than 100 digits" &
220             " in length & both integers cannot be of length 100" &
221             " this causes an overflow" & vbCrLf)
222         MessageBoxButtons.OK, MessageBoxIcon.Warning)
223     Else
224         ' perform addition
225         lblResult.Text = remoteInteger.Add(
226             txtFirst.Text, txtSecond.Text).TrimStart(zeros)
227     End If
228 End Sub
229
230 Private Sub btnSub_Click
231     ' subtracts two numbers input by user
232     ' make sure subtract_Click_Click() makes sure that
233     ' both numbers are at least 100 digits long
234     ' make sure hugeIntegers do not exceed 100 digits
235     ' but numbers can be of length 100, as long as they are
236     ' not used for subtraction
237     Dim result As String = remoteInteger.Subtract(
238         txtFirst.Text, txtSecond.Text).TrimStart(zeros)
239
240     If result = "" Then
241         lblResult.Text = ""
242     Else
243         lblResult.Text = result
244     End If
245 End Sub
246
247 ' click event for btnSubtract
248 Private Sub btnSubtract_Click(ByVal sender As Object,
249     ByVal e As System.EventArgs) Handles btnSubtract.Click
250     MessageBox.Show(
251         "First operand can be greater than second" & vbCrLf)
252 End Sub

```

Fig. 19.29 | Consuming the HugeInteger web service in Silverlight 1.1 Alpha. (Part 4 of 6.)

```

204         End Try
205     End If
206 End Sub ' btnLarger_Click
207
208 ' determines whether first number input is larger than the second
209 Private Sub btnLarger_Click(ByVal sender As System.Object,
210     ByVal e As System.EventArgs) Handles btnLarger.Click
211     ' ensure that HugeIntegers do not exceed 100 digits
212     If Not Integer.TryParse(txtFirst.Text, remoteInteger) Then
213         ' call web-service method to determine whether
214         ' first integer is larger than the second
215         If remoteInteger.Bigger(txtFirst.Text, txtSecond.Text) Then
216             lblResult.Text = txtFirst.Text.ToString(zeros) &
217                 " is larger than " & txtSecond.Text.ToString(zeros)
218         Else
219             lblResult.Text = txtFirst.Text.ToString(zeros) &
220                 " is not larger than " & txtSecond.Text.ToString(zeros)
221         End If
222     End If
223 End Sub ' btnLarger_Click
224
225 ' determines whether first number input is smaller than the second
226 Private Sub btnSmaller_Click(ByVal sender As System.Object,
227     ByVal e As System.EventArgs) Handles btnSmaller.Click
228     ' ensure that HugeIntegers do not exceed 100 digits
229     If Not Integer.TryParse(txtFirst.Text, txtSecond.Text) Then
230         ' call web-service method to determine if
231         ' first integer is smaller than second
232         If remoteInteger.Smaller(txtFirst.Text, txtSecond.Text) Then
233             lblResult.Text = txtFirst.Text.ToString(zeros) &
234                 " is smaller than " & txtSecond.Text.ToString(zeros)
235         Else
236             lblResult.Text = txtFirst.Text.ToString(zeros) &
237                 " is not smaller than " & txtSecond.Text.ToString(zeros)
238         End If
239     End If
240 End Sub ' btnSmaller_Click
241
242 ' determines whether the two numbers input are equal
243 Private Sub btnEqual_Click(ByVal sender As System.Object,
244     ByVal e As System.EventArgs) Handles btnEqual.Click
245     ' ensure that HugeIntegers do not exceed 100 digits
246     If Not Integer.TryParse(txtFirst.Text, txtSecond.Text) Then
247         ' call web-service method to determine if integers are equal
248         If remoteInteger.EqualTo(txtFirst.Text, txtSecond.Text) Then
249             lblResult.Text = txtFirst.Text.ToString(zeros) &
250                 " is equal to " & txtSecond.Text.ToString(zeros)
251         Else
252             lblResult.Text = txtFirst.Text.ToString(zeros) &
253                 " is not equal to " & txtSecond.Text.ToString(zeros)
254         End If
255     End If
256 End Sub ' btnEqual_Click

```

Fig. 19.29 | Consuming the HugeInteger web service in Silverlight 1.1 Alpha. (Part 5 of 6.)

```

248
249     * determines whether numbers input by user are too big
250     Private Function NumbersTooBig(ByVal first As String,
251     ByVal second As String) As Boolean
252     * display an error message if either number has too many digits
253     If (first.Length > 100) Or (second.Length > 100) Then
254     MessageBox.Show("HugeIntegers must be less than 100 digits",
255     "Error", MessageBoxButtons.OK, MessageBoxIcon.Information)
256     Return True
257     End If
258
259     Return False
260 End Function * NumbersTooBig
261
262 End Class

```

Fig. 19.29 | Consuming the HugeInteger web service in Silverlight 1.1 Alpha. (Part 6 of 6.)

The code in Fig. 19.29 uses the HugeInteger web service to perform computations with positive integers up to 100 digits long. You are already familiar with converting a Visual Basic Windows Forms application to Silverlight, so we focus our discussion on the web services concepts in this example.

Line 151 creates variable `remoteInteger` and initializes it with a proxy object of type `com.deitel.test.HugeInteger`. This variable is used in each of the application's event handlers to call methods of the HugeInteger web service. Lines 171–172, 183–184, 207, 224 and 240 in the various button event handlers invoke methods of the web service. Note that each call is made on the local proxy object, which then communicates with the web service on the client's behalf.

The user inputs two integers, each up to 100 digits long. Clicking a button causes the application to invoke a web method to perform the appropriate task and return the result. Note that client application HugeInteger cannot perform operations using 100-digit numbers directly. Instead the application creates `String` representations of these numbers and passes them as arguments to web methods that handle such tasks for the client. It then uses the return value of each operation to display an appropriate message.

Note that the application eliminates leading zeros in the numbers before displaying them by calling `String` method `TrimStart`, which removes all occurrences of characters specified by a `Char` array (line 154) from the beginning of a `String`.

19.11 Silverlight Demos, Games and Web Resources

In this section we provide links to, and descriptions of, several websites where you'll find Silverlight demos, games, controls, sample code and tutorials. For additional Silverlight resources (including tutorials, articles, blogs, books, sample chapters, community sites, FAQs, RSS feeds, podcasts, videos and more), visit the Deitel Silverlight Resource Center at www.deitel.com/silverlight.

silverlight.net/community/communitygallery.aspx

The Silverlight Gallery provides a large collection of Silverlight 1.0 and 1.1 sample applications. Check out the top-rated and recently added samples, or view the complete list. Each sample includes a star rating, a description and options for viewing and downloading the samples. Become a member to upload and share your Silverlight applications with the community.

www.hanselman.com/blog/SilverlightSamples.aspx

A collection of Silverlight sample applications (many overlap with Microsoft's Silverlight Gallery) compiled by Scott Hanselman, a Microsoft MVP.

community.netikatech.com/demos/

GOA WinForms demos from Netika Tech, available for Silverlight and Flash. GOA WinForms is an implementation of the .NET System.Windows.Forms library in Silverlight and Flash for developing RIAs. The simple demos include quick tours of GOA WinForms controls, a DataGrid, an Outlook-like calendar and a Visual Studio-like form designer.

www.andybeaulieu.com/Home/tabid/67/EntryID/73/Default.aspx

Silverlight Rocks! is a simple shooter game built with Silverlight 1.1. Using four buttons on your keyboard, you can turn the spaceship left or right, shoot and thrust forward. The goal is to destroy the asteroids. The author provides a walkthrough of how he wrote the game. The source code is available for download.

www.andybeaulieu.com/Home/tabid/67/EntryID/75/Default.aspx

Destroy All Invaders is a shooter game built with Silverlight 1.1. Select a location from the drop-down list (options include rural upstate New York, Microsoft's Redmond campus and Las Vegas, to name a few) or enter a specific address. The game brings up a satellite image of the location and an animated helicopter. The point of the game is to shoot and destroy the UFOs. The author provides a walkthrough of how he wrote the game. The source code is available for download.

www.blurosegames.com/brg/drpopper/default.html

Dr. Popper Silverlight Edition by Bill Reiss of Blue Rose Games is written for Silverlight 1.1. The game consists of multiple colored bubbles arranged on a 10-bubble by 8-bubble board. You can remove the bubbles in groups of two or more, gaining more points for bigger groups. The source code is available for download.

www.aisto.com/Roeder/Silverlight/Monotone/Default.aspx

Monotone is an animated graphics demo built for Silverlight (using C#) and MP3 audio. Download the source code at www.aisto.com/Roeder/Silverlight/.

www.aisto.com/Roeder/Silverlight/Inplay/Default.aspx

InPlay is an in-browser audio and video player. The demo includes stunning audio and video, and you can use the controls to adjust the volume and position. Source code for InPlay is available at www.aisto.com/Roeder/Silverlight/.

zerogravity.terralever.com/

Zero Gravity is an adventure game, created by Terralever using Silverlight and C#. The game features animation and audio. Using your keyboard controls, the goal is to get Lieutenant Bennett back to his spaceship safely, jumping between blocks, teleports, switches and more.

silverlight.net/samples/1.0/Sprawl/default.html

Sprawl, written for Silverlight, is a tile-capture game in which you play against the computer. The goal is to capture more tiles than the computer without paving over tiles you have already captured.

cosmik.members.winisp.net/BubbleFactory/

The Bubble Factory game, built with Silverlight, is a simple animated game in which you use keyboard controls to move the bubble dropper left or right and to drop the bubbles. The key is to align three bubbles of the same color (horizontally, vertically or diagonally) to make them explode.

silverlight.net/samples/1.1/chess/run/default.html

A simple game of chess built with Silverlight 1.1.

microsoft.blognewschannel.com/archives/2007/06/29/barrel-o-silverlight-games/

The Inside Microsoft blog entry entitled "Barrel O' Silverlight Games" includes links to several Silverlight games including Chess, Zero Gravity, Sprawl, Destroy All Invaders, Digger and more.

792 Internet & World Wide Web How to Program

silverlightrocks.com/community/blogs/silverlight_games_101/default.aspx

A tutorial entitled “Silverlight Games 101: Beginning game programming in Microsoft Silverlight 1.1 using C#” by Bill Reiss and Silverlight Rocks! Topics include the Zero Gravity game, loading XAML dynamically, adding thrusts, a better game loop, keyboard input, creating a game loop and drawing a sprite. All of the source code for the tutorial is available for download.

blogs.msdn.com/tims/default.aspx

Microsoft’s Tim Sneath blogs about Silverlight and other Microsoft technologies. He includes links to 50+ Silverlight samples, information about the latest Silverlight releases and other Silverlight news.

www.junkship.org:8000/silverlightdemo/

The Amazon Search Visualization demo, built with Silverlight. Click the “New Search” button, then enter the title of the book or author for which you would like to search. Images of each book and related books appear on the screen. Click the green button on the image to get the book details (including title, author(s), reviewer rating, lowest new price and lowest used price). You will also see a visual presentation of book covers for related books. Click the red button on the book cover to close that item. Click the “Clear All” button to search for a new book.

dnnsilverlight.adevwebserver.com/Samples/SilverlightVideo/tabid/55/Default.aspx

A Silverlight Video module for DotNetNuke allows you to embed a video player in your DotNetNuke site. Check out the demo to view a video in a web page or to view the video full-screen. The site includes installation and configuration instructions.

www.chriscavanagh.com/Chris/Silverlight/Physics2D-1/TestPage.html

A 2-D Physics Engine has numerous platforms. Click the “Drop Wheels” button to drop tires from the top of the page onto the varying platforms to see which direction they will roll. Click the “Move Platforms” button and “Drop Wheels” to try again.

dev.aol.com/mail

The AOL Social Mail Gadget, built with Silverlight, gives AOL mail users easy access to email, IM, photos and video and more with just one click. It also allows you to set up an “A-List” of your most important contacts so you are aware when they are online, when a message from someone on the list is received and more.

mlb.mlb.com/media/video.jsp

Check out a sample of a Silverlight video player embedded in a Major League Baseball web page. You can pause and rewind the video and adjust the volume. A link is provided so you can link to the video from your website.

silverlight.net/samples/1.0/Grand-Piano/default.html

A Grand Piano application built with Silverlight includes audio and animation. Click on the key with your mouse to play a note.

www.telerik.com/products/silverlight/overview.aspx

RadControls for Microsoft Silverlight help you build RIAs without using JavaScript or XAML coding. Features include layouts, animation effects, integration with ASP.NET Ajax and more. Check out the demos to see the features, functionality, appearance and more.

blogs.msdn.com/cbowen/archive/2007/07/30/controls-and-control-libraries-for-silverlight.aspx

The blog entry entitled “Controls and Control Libraries for Silverlight” by Microsoft’s Chris Bowen, provides links to some of the reusable Silverlight controls and libraries that allow you to develop Silverlight applications faster and more efficiently. You’ll also find links to samples and tutorials.

silverlight.net/QuickStarts/BuildUi/CustomControl.aspx

The tutorial “How to Create Custom Silverlight Controls” discusses the control UI and object model, starting from the Silverlight Class Library Project, defining the UI, getting object references,

adding properties and events for control customization, testing your control and shadowing inherited properties.

silverlight.net/Learn/Learnvideo.aspx?video=207

The video tutorial “How to Build a Silverlight Control” by Jesse Liberty, shows you how to create an HTML application that interacts with a Silverlight control.

www.codeplex.com/Project/ProjectDirectory.aspx?ProjectSearchText=silverlight

CodePlex, Microsoft’s open source project hosting website, includes a collection of 14 open source Silverlight projects including iTunes 2.0, Dynamic Silverlight Samples, Silverlight 1.0 JavaScript Intellisense, Silverlight Playground, Balder, Silverlight Audio Player and more. Each project includes a description of the project, a demo and the source code.

www.aisto.com/Roeder/Silverlight/

Lutz Roeder’s Silverlight page provides several sample applications including Monotone (www.aisto.com/Roeder/Silverlight/Monotone/Default.aspx), a graphics application written in XAML and C#; Digger (www.aisto.com/Roeder/Silverlight/Digger/Default.aspx), a clone of the Boulderdash game, written in C#; and Inplay (www.aisto.com/Roeder/Silverlight/Inplay/Default.aspx?Audio=play:false&Video=source:http://download.microsoft.com/download/2/C/4/2C433161-F56C-4BAB-B8C5-B8C6F240AFCC/SL_0410_448x256_300kb_2passCBR.wmv), an audio and video player that can be embedded in a web page, built with C#. Download demos of each application and get the source code.

Summary

Section 19.1 Introduction

- Silverlight is Microsoft’s RIA platform. It is designed to complement Ajax and other RIA technologies, such as Adobe Flash and Flex, Sun’s JavaFX and Microsoft’s own ASP.NET Ajax.
- Silverlight currently runs as a browser plug-in for Internet Explorer, Firefox and Safari on recent versions of Microsoft Windows and Mac OS X.
- Developers from the Mono project are developing an open-source implementation of Silverlight for Linux distributions called Moonlight.
- At the time of this writing, Silverlight is currently available in version 1.0 Release Candidate and version 1.1 Alpha Refresh.

Section 19.2 Platform Overview

- Silverlight applications consist of a user interface described in Extensible Application Markup Language (XAML) and a code-behind file (or files) containing the program logic.
- XAML is Microsoft’s XML vocabulary for describing user interfaces.
- Silverlight 1.0 focuses primarily on media and supports programming only in JavaScript.
- Microsoft provides a service called Silverlight Streaming that allows you to distribute video-based Silverlight applications for free.
- When Silverlight 1.1 is released, computers with Silverlight 1.0 will automatically be upgraded. This could immediately make Silverlight 1.1 a widespread platform for RIA development.
- Silverlight 1.1’s key benefit is that it adds an implementation of the .NET runtime, allowing developers to create Silverlight applications in .NET languages.
- Microsoft plans to implement a built-in set of controls in a future release of Silverlight 1.1.

- Version 1.1 provides a substantial performance improvement over 1.0 because .NET code is compiled by the developer then executed on the client, unlike JavaScript, which is interpreted and executed on the client at runtime.

Section 19.3 Silverlight 1.0 Installation and Overview

- We developed our Silverlight 1.0 application using Microsoft's Expression Blend 2, a WYSIWYG editor for building XAML user interfaces.

Section 19.4 Creating a Movie Viewer for Silverlight 1.0

- To create the project in Expression Blend, open Expression Blend and select **New Project** in the **Project** tab. To create a Silverlight 1.0 application, select **Silverlight Application (JavaScript)**.

Section 19.4.1 Creating a User Interface In XAML Using Expression Blend

- The root element of the XAML file is a **Canvas** element. A **Canvas** element acts as a container for other user interface elements and controls their position.
- The parent **Canvas** element is created when you create a new Silverlight project in Expression Blend. The parent **Canvas** has a default **Name of Page**, **Width** of 640 px and **Height** of 480 px.
- An element's **Name** attribute provides an ID to access the element programmatically.
- An element's properties can be edited in the **Properties** panel.
- Additional **Canvas** elements can be created in Expression Blend using the toolbar's **Canvas** tool.
- The XAML can be manually edited by selecting **XAML** in Expression Blend's **View** menu.

Section 19.4.2 Using Storyboards

- The **Storyboard** element allows you to define animations.
- In Expression Blend, you can create a **Storyboard** by opening the **Open, create or manage Storyboards** panel and clicking the **Create new Storyboard** button. Selecting the **Create as a Resource** checkbox enables you to start the **Storyboard** from anywhere in the application's JavaScript.
- A **Storyboard** must have a target object.
- Expression Blend provides the **Gradient brush** tool to visually create and modify gradients.

Section 19.4.3 Creating Controls

- You can create a **TextBlock** element using Expression Blend's **TextBlock** tool.
- Use the **Solid color brush** in the **Brushes** inspector to set a solid color.
- You can adjust the text size in the **Text** inspector.
- A **Canvas** element can be a child of another **Canvas** element.
- Double-click a **Canvas** element with the **Selection** tool to activate it.
- The **Image** element's **Source** attribute points to an image file. You can select the **Image** tool by clicking the **Asset Library** button, checking **Show All**, and selecting **Image**.
- The user's cursor will change to a hand when the cursor is moved over a **Canvas** if its **Cursor** property is set to **Hand** in the **Common Properties** inspector.
- Set the **RadiusX** and **RadiusY** to give a **Rectangle** rounded corners. These properties are located in the **advanced properties** section of the **Appearance** inspector.
- Use the **Pen** tool to draw a **Path**. The **Path** element allows you to draw shapes that include curves and arcs, but here we are just using it to draw simple lines.
- You can set the **StrokeThickness**, **StrokeEndLineCap** and **StrokeStartLineCap** properties of a **Path** in the **Appearance** inspector.

- One element appears on top of another if it appears after the other element in the **Objects and Timeline** inspector. You can also specify the *z*-order of elements using an object's **ZIndex** attribute. Higher ZIndex integer values position the element closer to the foreground.
- The **MediaElement** allows you to include video and/or audio. To access the **MediaElement** tool, click the **Asset Library** button, check **Show All** and select **MediaElement**.
- The **MediaElement**'s **Source** attribute points to the source video file.
- Expression Blend 2 August Preview does not currently have a user interface to set event handlers, so you must manually set them in the XAML.
- **Storyboard** attribute **Completed** registers an event that is triggered when an animation completes.
- The **MouseDown** attribute registers an event that is triggered when the user left-clicks on the element.
- **MediaElement** attribute **MediaOpened** registers an event that is triggered when a video opens.
- When a **MediaElement** is loaded, it will begin playing the movie. To change this, set its **AutoPlay** attribute to **false**.
- **MediaElement** attribute **MediaEnded** registers an event that is triggered when a video has reached the end.
- An alternative to registering event handlers in the XAML is to register event handlers in the JavaScript code. This has two key advantages—it keeps the application's logic separate from the application's user interface, and it allows you to add and remove event listeners dynamically.
- When registering an event in JavaScript using the **addEventListener** method, store the method's return value, so you can remove the event listener using the **removeEventListener** later.

Section 19.4.4 Using JavaScript for Event Handling and DOM Manipulation

- The JavaScript code-behind file, **Page.xaml.js**, defines the event handlers for the various elements in the XAML.
- A **Canvas**'s **Loaded** event is triggered when the **Canvas** finishes loading.
- You can create a reference to a XAML element using the sender's **findName** method.
- Every event handler receives **sender** and **eventArgs** parameters. The **sender** parameter is a reference to the element with which the user interacted, and the **eventArgs** parameter passes information about the event that occurred.
- Method **getHost** returns a reference to the Silverlight plug-in so you can access its properties.
- The plug-in's **onFullScreenChange** event occurs when the application switches to or from full-screen mode.
- A **Storyboard**'s **begin** function starts the **Storyboard**.
- A **Storyboard**'s **Completed** event occurs when the animation completes.
- The **MediaElement**'s **position.Seconds** attribute contains the media's elapsed time in seconds.
- In the **Canvas.Left** attribute, **Left** is a dependency property of **Canvas**, meaning that the **Left** value is relative to the **Canvas**. To access a dependency property, enclose the attribute name in quotes and square brackets, as in `element["attributeName"]`.
- The plug-in's **fullScreen** attribute specifies whether the application is in full-screen mode.
- **MediaElement** property **volume** is a value between 0 (muted) and 1 (full volume).

Section 19.5 Embedding Silverlight in HTML

- Expression Blend generates an HTML wrapper named **Default.html** for your Silverlight application when you first create the Silverlight project.

- You can embed a Silverlight application into an existing HTML file by including the scripts, the `silverlightHost` style class and the `SilverlightControlHost` div from `Default.html`.
- You can adjust the width and height of your application by changing the width and height attributes of the `silverlightHost` style class.

Section 19.6 Silverlight Streaming

- Microsoft's Silverlight Streaming (silverlight.live.com) enables individuals and businesses to share video content online. You can easily embed Silverlight applications that are hosted by this service into your web pages.

Section 19.7 Silverlight 1.1 Installation and Overview

- Silverlight 1.1 uses a lightweight version of the .NET CLR (Common Language Runtime) in the browser plug-in. This allows you to program Silverlight applications in many .NET languages.
- Silverlight 1.1 applications use the .NET CLR's just-in-time (JIT) compiler to compile the code to machine language, providing significant performance improvements over the interpreted JavaScript used in Silverlight 1.0 and Ajax.
- We developed our Silverlight 1.1 applications using Microsoft Expression Blend 2 and Microsoft Visual Studio 2008. The **Silverlight Tools Alpha for Visual Studio** enable you to create a Silverlight 1.1 Alpha Refresh project.

Section 19.8 Creating a Cover Viewer for Silverlight 1.1 Alpha

- To create a Silverlight 1.1 Alpha Refresh project, open Visual Studio 2008 and select **New Project** in the **File** menu.
- A Silverlight 1.1 Alpha Refresh project will initially contain a XAML file, `Page.xaml`, a code-behind file, `Page.xaml.vb`, `Silverlight.js` and the HTML wrapper, `TestPage.html`.
- The `x:Class` attribute specifies the class that contains the event handlers.
- The `InitializeComponent` method in the autogenerated `Page.g.vb` file, takes any XAML elements that have an `x:Name` attribute and uses the `FindName` method to map each element to a variable of the same name.
- Note that the `BrowserHttpRequest` object does not currently support cross-domain requests, so your application and its resources must be located on the same server.
- Use the `HtmlPage` element to find the `AbsolutePath` of the page.

Section 19.9 Building an Application with Third-Party Controls

- Though Silverlight 1.1 Alpha Refresh does not yet include pre-built controls, a number of third-party control libraries have been created.
- Netika's GOA WinForms library implements .NET's `System.Windows.Forms` library for both Silverlight and Flash.
- To create a GOA WinForms Silverlight application, open Visual Studio 2008 and create a new project. Select **Visual Basic**, then **GOA WinForms VB Application in My Templates**.
- For a GOA WinForms project, the Visual Basic code-behind file is located in `MyForm.vb`.
- To convert a Visual Basic desktop application to a Silverlight application using GOA WinForms, copy code from the user interface and code-behind files into `MyForm.vb`.
- You may see errors because not every property of the Windows Form controls has been implemented in GOA WinForms.

- Some of the controls function slightly differently, as GOA WinForms is not an exact replica of the standard Windows Forms controls.

Section 19.10 Consuming a Web Service

- A proxy class (or proxy) is generated from a web service's WSDL file and enables the client to call web methods over the Internet. The proxy class handles all the details of communicating with the web service.
- When you add a web reference to the Silverlight project, Visual Studio will generate the appropriate proxy class. You will then create an instance of the proxy class and use it to call the web service's methods.
- At this time, a Silverlight application that invokes a web service must reside on the same domain as that web service, because Silverlight 1.1 does not yet allow for cross-domain requests.
- Add the web reference by clicking the **Add Reference** button (Fig. 19.27).

Terminology

addEventListener method	MouseDown attribute
AutoPlay attribute of MediaElement element	naturalDuration attribute of a MediaElement
BrowserHttpWebRequest object	NET Common Language Runtime (CLR)
C#	onFullScreenChange event (Silverlight plug-in)
Canvas element	Opacity attribute
Canvas.Left attribute	Path element
code-behind file	position.Seconds attribute of MediaElement
Collapsed value of Visibility	proxy class for a web service
Completed attribute of Storyboard element	Rectangle element
dependency property	removeEventListener method
Ellipse element	sender parameter
eventArgs parameter	Silverlight
Expression Blend 2	Silverlight Document Object Model (DOM)
Fill attribute	Silverlight plug-in
FindName method of the Silverlight 1.0 plug-in	Silverlight Screaming
FindName method of the Silverlight 1.1 plug-in	Silverlight Tools Alpha for Visual Studio
fullScreen attribute of Silverlight plug-in	Source attribute of Image
getHost method	Source attribute of MediaElement
Gradient brush tool	Storyboard element
Hand value of Cursor attribute	Stroke attribute
Height attribute	TextBlock element
HTML wrapper	uri
HttpPage element	Visibility attribute of Canvas element
Image element	Visual Basic
IntelliSense	volume attribute of MediaElement element
JavaScript	Web reference selection and description
just-in-time (JIT) compiler	Width attribute
Loaded event of Canvas element	Windows Media Video (WMV)
Manifest for Silverlight Streaming	Windows Presentation Foundation (WPF)
MediaElement element	Windows Presentation Foundation Everywhere (WPF/E)
MediaEnded attribute of MediaElement	WMV (Windows Media Video)
MediaOpened attribute of MediaElement	

x:Class attribute of Canvas element
 x:Name attribute
 XAML (Extensible Application Markup Language)

XamlReader object
 ZIndex attribute of Canvas
 z-order

Self-Review Exercises

19.1 Fill in the blanks in each of the following statements:

- A(n) _____ is the parent element of a XAML file.
- A(n) _____ is used to embed video and audio in a Silverlight application.
- Visual Basic's _____ class can be used to parse RSS in Silverlight.
- The _____ event is triggered by clicking an element with the left mouse button.
- Animations are described using the _____ XAML element.
- _____ allows you to distribute Silverlight applications containing audio and video for free.
- _____ allows you to visually edit XAML.
- You can use an object's _____ attribute to specify the z-order of elements.
- A MediaElement's _____ attribute determines whether the media will start playing immediately after it has loaded.
- The Silverlight plug-in's _____ method retrieves an element of a specific name.
- In Silverlight 1.1, the _____ attribute specifies the Class that contains the XAML elements' event handlers.

19.2 State whether each of the following is true or false. If false, explain why.

- The Silverlight browser plug-in runs on IE and Firefox on Windows XP and Vista, and Safari and Firefox on Mac OS X.
- Silverlight applications can be embedded in an existing HTML file.
- Silverlight requires server-side software.
- You can program Silverlight event handlers in XAML.
- Silverlight 1.1 supports programming in C#, Visual Basic, and other .NET languages.
- Silverlight applications can run in full-screen mode.
- The `BrowserContextHost` object allows for cross-domain requests.
- A Silverlight application must have at least one Canvas element.
- There is no way to implement a timer in Silverlight 1.0.
- You can hide an element only by setting its `Opacity` to 0.

Exercises

19.3 Add mouse-over and mouse-down graphics for the controls in the Movie Viewer, to improve the feel of the user interface. To do this, add the `MouseEnter`, `MouseLeave`, and `MouseDown` events in the XAML and corresponding event handlers in the JavaScript code behind. An example of how your solution may look is available at test.dnet.com/examples/1w3http4/silverlight/MovieViewer2/index.html.

19.4 Enhance the book-cover viewer application so that when the user switches covers, the new image zooms in instead of instantly appearing. Do this by adding `nextNextImage` and `prevPrevImage` elements that are initially hidden. An example of how your solution may look is available at test.dnet.com/examples/1w3http4/silverlight/CoverViewer2/index.html.



Adobe Dreamweaver CS3

OBJECTIVES

In this chapter you will learn:

- To use Dreamweaver CS3 effectively.
- To develop web pages in a visual environment.
- To insert images and links into web pages.
- To create XHTML elements such as tables and forms.
- To insert scripts into Dreamweaver pages.
- To use the Spry framework to create richer, more dynamic web applications.
- To use Dreamweaver's site-management capabilities.



*We must select the illusion
which appeals to our
temperament, and embrace
it with passion, if we want to
be happy.*

—Cyril Connolly

*The symbolic view of things
is a consequence of long
absorption in images. Is sign
language the real language of
Paradise?*

—Hugo Ball

*What you see is what you get
(WYSIWYG).*

—Anonymous

*All human knowledge takes
the form of interpretation.*

—Walter Benjamin

Outline

- 20.1 Introduction
- 20.2 Adobe Dreamweaver CS3
- 20.3 Text Styles
- 20.4 Images and Links
- 20.5 Symbols and Lines
- 20.6 Tables
- 20.7 Forms
- 20.8 Scripting in Dreamweaver
- 20.9 Spry Framework for Creating Ajax Applications
- 20.10 Site Management
- 20.11 Web Resources

Summary | Terminology | Self-Review Exercises | Exercises

20.1 Introduction

This chapter presents Adobe's *Dreamweaver CS3*, perhaps the most popular visual HTML editor. A fully functional, 30-day trial version of Dreamweaver is available for download at www.adobe.com/cfusion/tdrc/index.cfm?product=dreamweaver. Please download and install the software before studying this chapter.

Using Dreamweaver, you can easily perform many of the tasks you learned in previous chapters. You can insert and edit text, as well as create more complex XHTML elements, such as tables, forms, frames and much more. In addition, this latest version of Dreamweaver now enables you to develop Ajax applications with Adobe's Spry framework.

20.2 Adobe Dreamweaver CS3

Upon starting, Dreamweaver displays the default **Start Page**, which offers various options, such as **Open a Recent Item**, **Create New** and **Create from Samples** (Fig. 20.1). For example, you can click the **HTML** option under the **Create New** heading to open a blank page in the default viewing mode (Fig. 20.2). Dreamweaver is a **WYSIWYG** (**What You See Is What You Get**) editor. Unlike editors that simply display XHTML code, Dreamweaver renders XHTML elements much as a browser would, using the WYSIWYG screen. This functionality enables you to design your web pages as they will appear on the web.

We will now recreate the book's first XHTML example (Fig. 4.1) using Dreamweaver. To see a more detailed list of options for creating new files, create a new document by selecting **New...** from the **File** menu. In the **New Document** dialog, select the **Blank page** tab from the leftmost column, and **HTML** from the **Page Type**: list (Fig. 20.3). By default, Dreamweaver's **DocType** (in the lower-right corner) is set to **XHTML 1.0 Transitional**. Select the drop-down **DocType** menu and select **XHTML 1.0 Strict**—this will cause Dreamweaver to generate XHTML-compliant code. In the **Layout**: list, make sure **<none>** is selected. Click the **Create** button to open the new document.

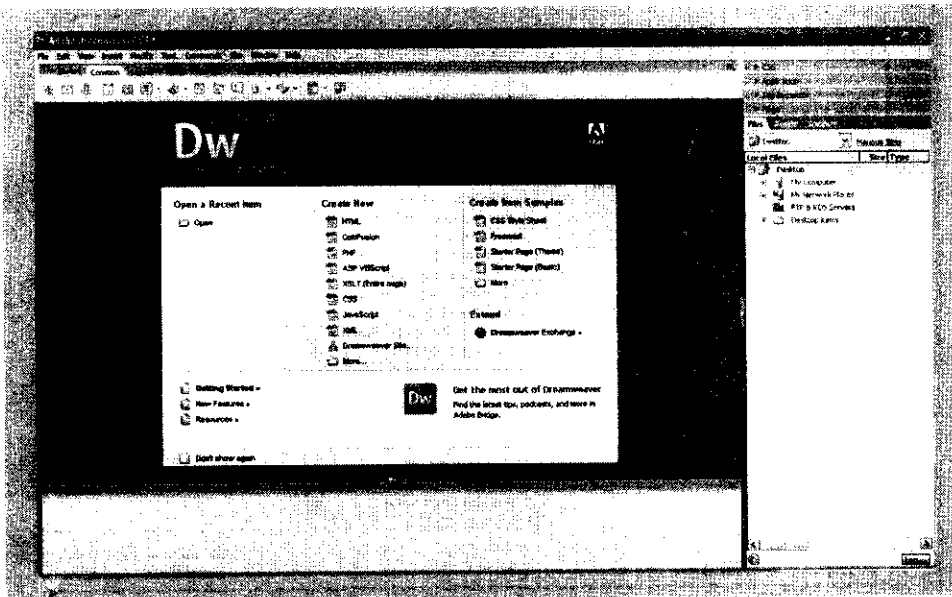


Fig. 20.1 | Dreamweaver Start Page.

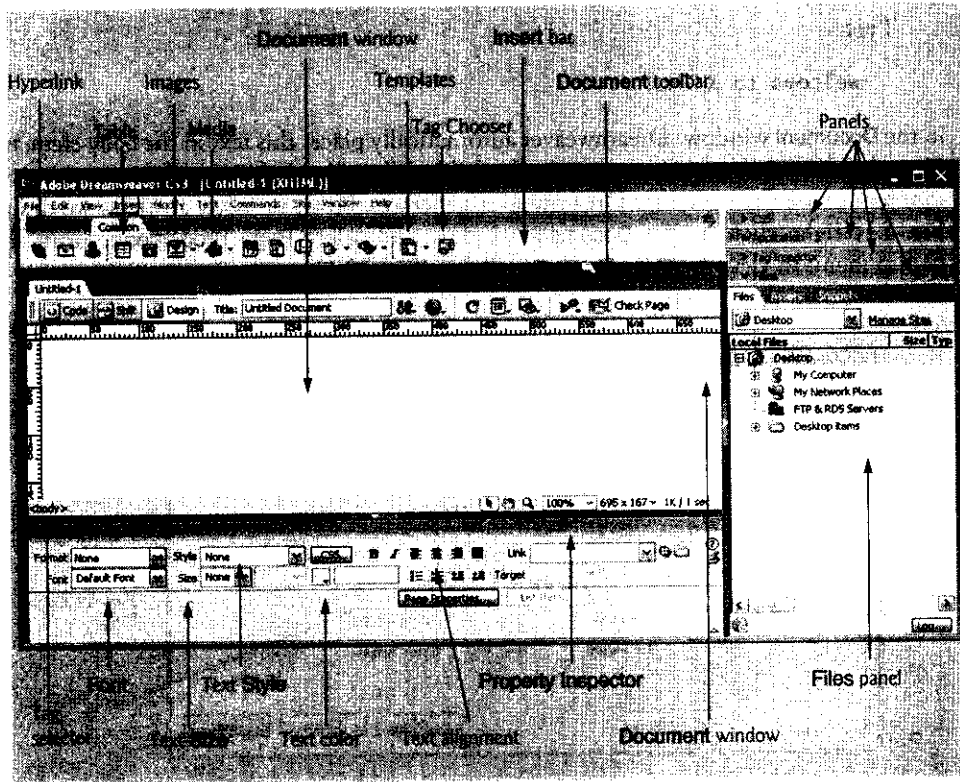


Fig. 20.2 | Dreamweaver editing environment.

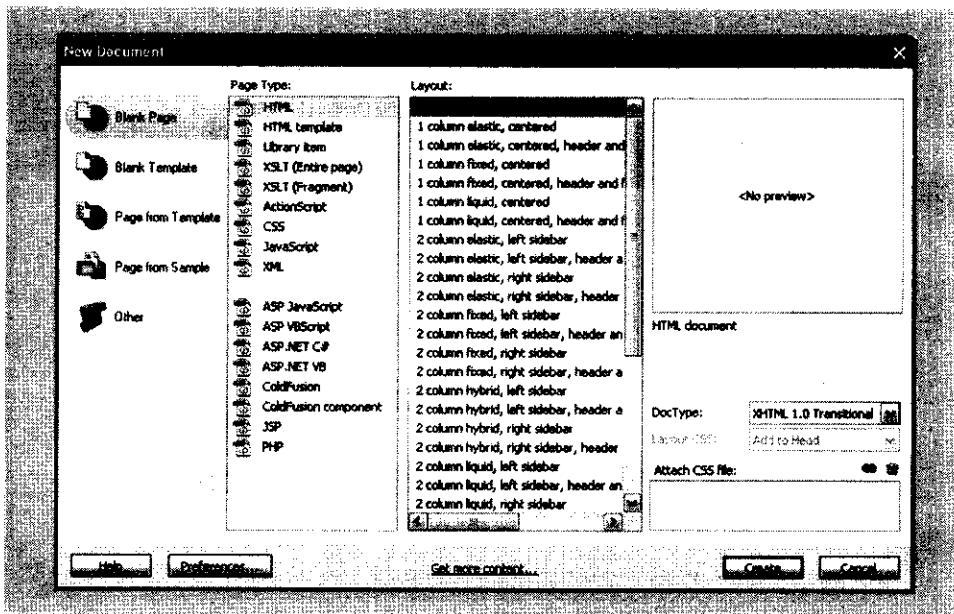


Fig. 20.3 | New Document dialog.

Type

Welcome to XHTML!

in the **Document** window. Dreamweaver automatically places this text in the body element. Note that XHTML tags are not currently visible. We will switch to an alternate view in a moment to see the code that Dreamweaver generates. Now, to insert a title as we did in Fig. 4.1, right click in the **Document** window and select **Page Properties...** from the popup menu to view the **Page Properties** dialog (Fig. 20.4).

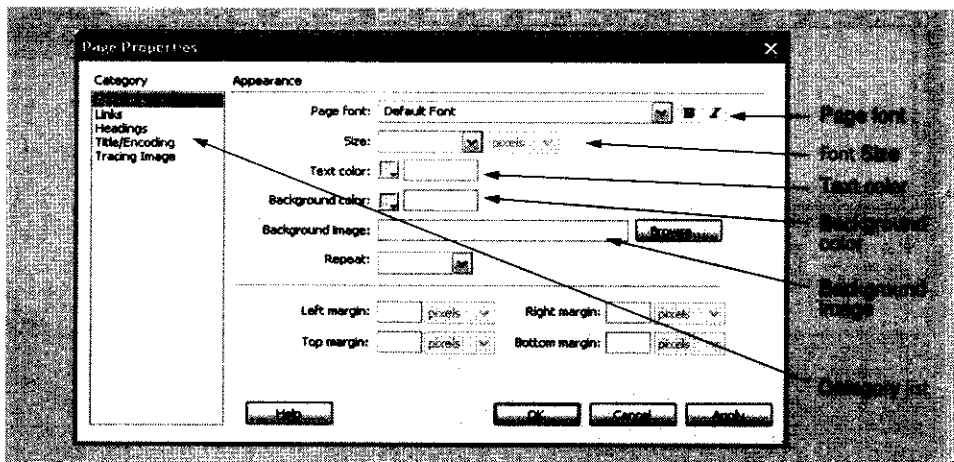


Fig. 20.4 | Page Properties dialog.

The **Category list** lets the user select a set of properties to view. Select **Title/Encoding** from the **Category list** and enter **Internet and WWW How to Program** into the **Title** field. Clicking **OK** inserts a `<title>` element with the corresponding title text inside the head element of your XHTML code. [Note: You can also create a `<title>` by entering text directly into the Document title box (Fig. 20.6).] You now have a representation of the code in Fig. 4.1 in the WYSIWYG display (Fig. 20.5).

Though you have been editing using the WYSIWYG display, remember that you are still programming in XHTML. To view or edit the XHTML that Dreamweaver generated, you must switch from **Design view**, the mode you are currently working in, to **Code view**. To do so, click the **Code button** in the **Document toolbar** (Fig. 20.6). Note that Dreamweaver automatically color-codes XHTML to make viewing easier (Fig. 20.7). The tag names, attribute values and page text are all displayed in different colors. The code-coloring scheme can be accessed (and modified) by selecting **Preferences...** from the **Edit** menu and clicking **Code Coloring** in the **Category list**.

To save your file, click **Save** in the **File** menu or press `<Ctrl>-S`. The **Save As** dialog will appear, allowing you to specify a filename, type and location (Fig. 20.8). Create a folder in your C: drive named **Dreamweaver sites**. Type `main` into the **File name** field and select **HTML Documents** as the file type. Dreamweaver adds an `.html` filename extension if no extension is specified.

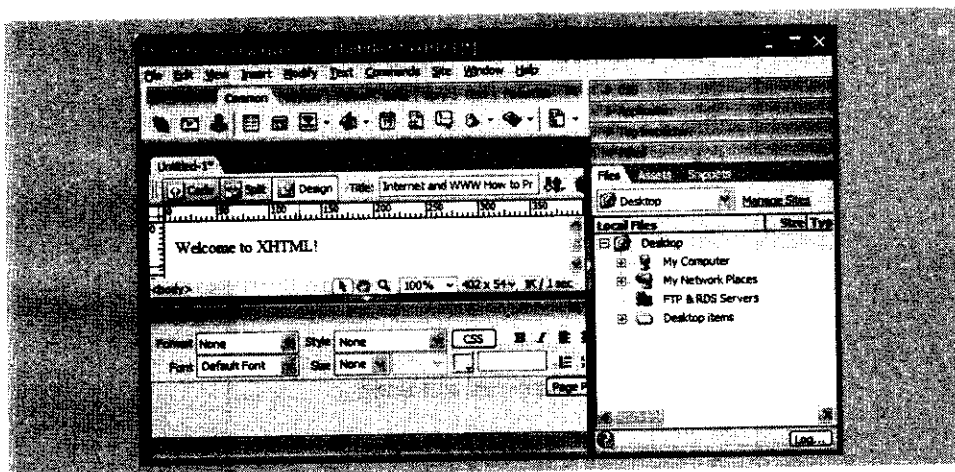


Fig. 20.5 | Example of Fig. 4.1 in Dreamweaver.

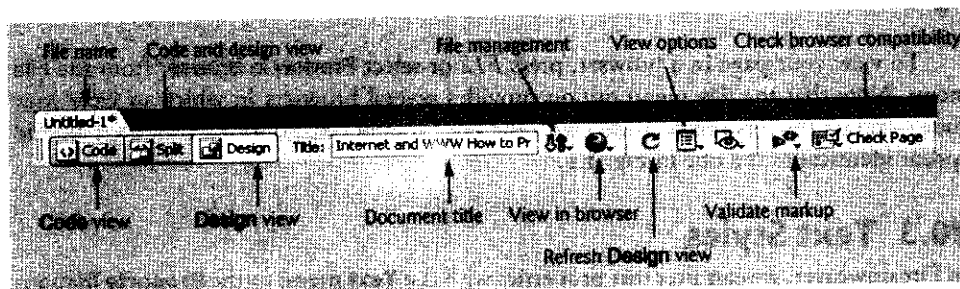


Fig. 20.6 | Document toolbar.

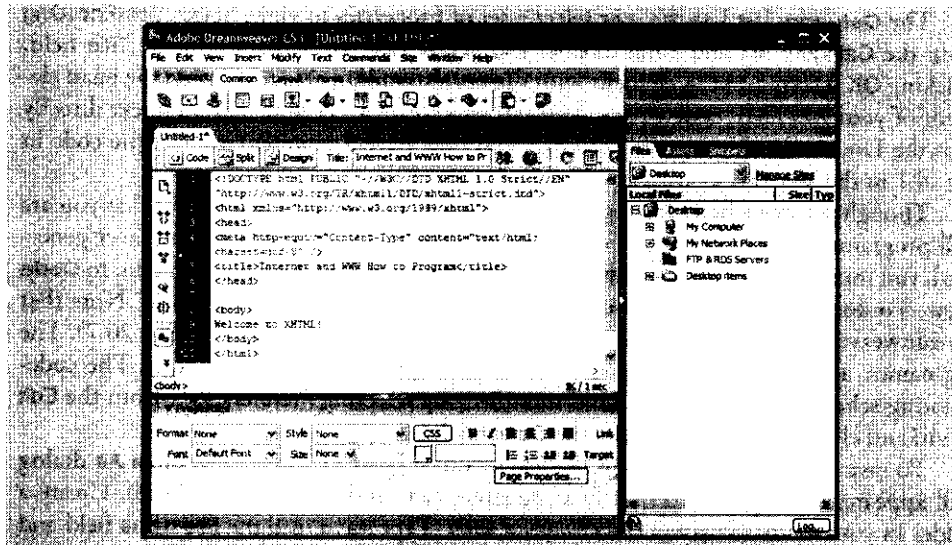


Fig. 20.7 | Code view.

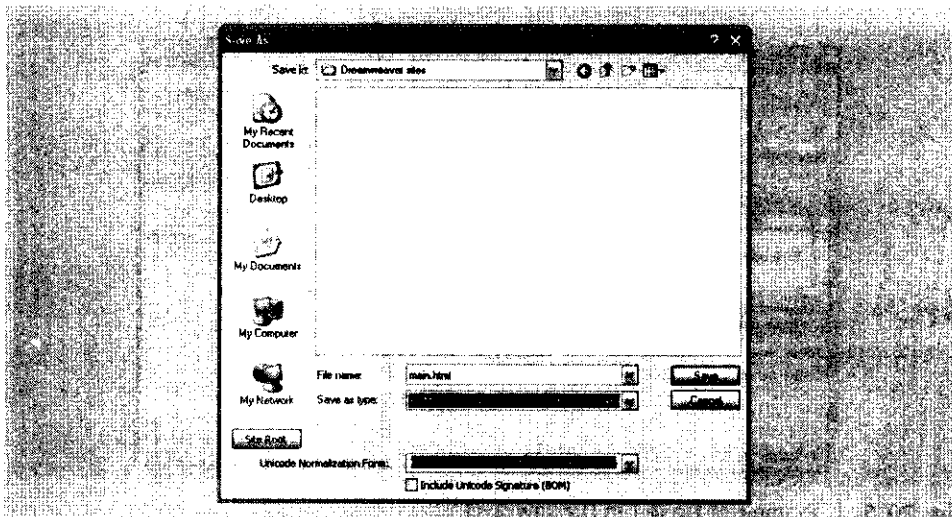


Fig. 20.8 | Save As dialog.

To view your page in a browser, press *F12* or select **Preview in Browser** from the **File** menu. Note that the **File** menu option provides several browsers in which to view your code—more browsers can be added with the **Edit Browser List...** option. Your page should appear identical to the one in Fig. 4.1.

20.3 Text Styles

In Dreamweaver, we can alter text properties with the **Text** menu or the **Property Inspector** (Fig. 20.2). Using these tools, we can quickly apply heading tags (<h1>, <h2>, etc.), list

tags (,) and several other tags used for styling text. Text can also be aligned left, right or centered, resized, indented and colored.

Create a new document, switch back to **Design** view and type the text, as shown in the screen capture of Fig. 20.9, into the **Document** window. Drag the mouse to highlight one line at a time and select the corresponding heading tag from the **Format** pull-down menu in the **Property Inspector**. Then, highlight all the text by pressing <Ctrl>-A, and click the align center button in the **Property Inspector**. The resulting XHTML produced by Dreamweaver is shown in Fig. 20.9.

As you can see, Dreamweaver is prone to produce somewhat inefficient code. In this case, for example, using Cascading Style Sheets (CSS) to center the text would have been more efficient. At the end of this section, we discuss how to integrate CSS into your web page without having to edit the XHTML in **Code** view.

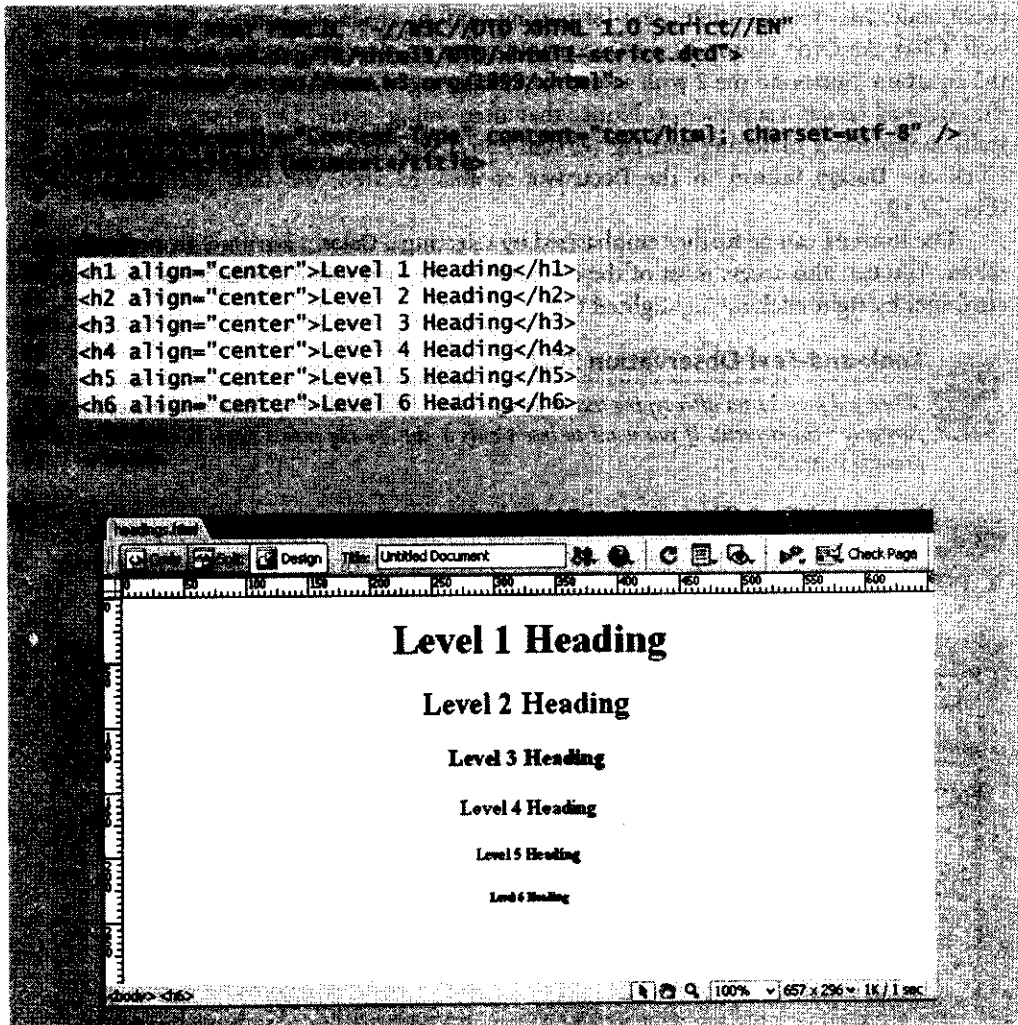


Fig. 20.9 | Applying heading tags and centering using Dreamweaver.



Software Engineering Observation 20.1

Dreamweaver uses text-manipulation techniques that sometimes produce inefficient code. Make sure to check the code often to know exactly the kind of XHTML Dreamweaver is producing. Thorough knowledge of a page and what XHTML elements are present is necessary for advanced scripting.

Dreamweaver is capable of much more extensive text formatting, such as creating mathematical formulas. For example, type

`e = mc2`

into a new WYSIWYG window, then highlight the text. You can now change the formatting of the equation by selecting **Style** from the **Text** menu, and selecting **Code**. The **Code** option applies a code element to the highlighted text, which designates formulas or computer code. Many other useful text-formatting options are located in the **Text** menu, as well. Click the **Code** button in the **Document** toolbar to view the code, and find the 2 in the equation. Surround the 2 with a `^{...}` tag. The `^{...}` tag formats the enclosed text as a superscript. Notice that after typing `^{`, Dreamweaver automatically completes a matching end tag (`}`) after you have entered the `</>` characters. Click the **Design** button in the **Document** toolbar to view the fully formatted text (Fig. 20.10).

The formula can be further emphasized by selecting a **Color...** attribute from the **Text** menu. You can also access most of the elements in the **Text** menu (though not the color attribute) by right clicking highlighted text.



Look-and-Feel Observation 20.1

*When you press Enter after typing text in Design view, Dreamweaver encloses that text in a new paragraph (p) element. If you want to insert only a `
` tag into a page, hold Shift while pressing Enter.*



Look-and-Feel Observation 20.2

You can manipulate the properties of almost any element displayed in the Dreamweaver window by right clicking an element and selecting its properties from the menu that pops up.

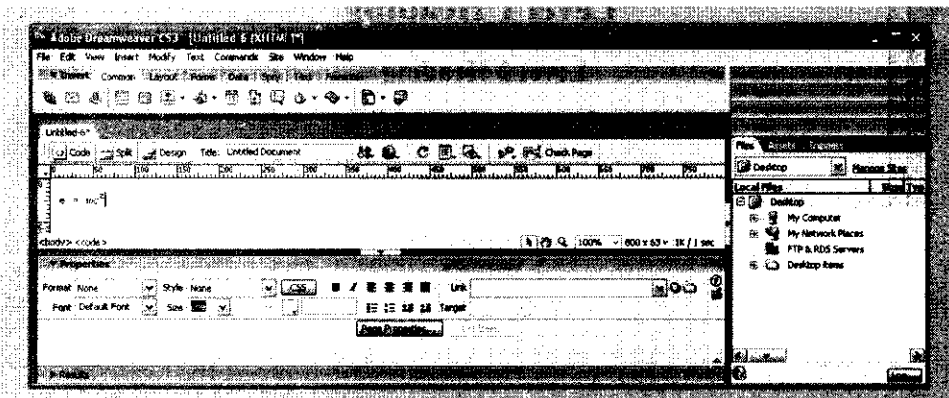


Fig. 20.10 | Styling text using code and sup elements.

The **Property Inspector** is also useful for creating lists. Try entering the contents of a shopping list, as shown in Fig. 20.11, and applying the **Unordered List** style to the list elements. Apply an h2 element to the title of the list.

Select **List** from the **Text** menu for more list-related tags, such as the definition list (<dl>). There are two list elements in a definition list—the defined term (<dt>) and the definition data (<dd>). Figure 20.12 shows the formatting produced by a definition list and the code Dreamweaver uses to produce it.

To apply the definition list as shown, select **Definition List** from the **List** submenu of the **Text** menu. In the **Document** window, type the first term you want to define. When you press *Enter*, Dreamweaver changes the style to match that of a definition. Pressing *Enter* again lets you enter another defined term. The bold style of the defined terms is applied by clicking the **Bold** button in the **Property Inspector**, which applies the strong element.

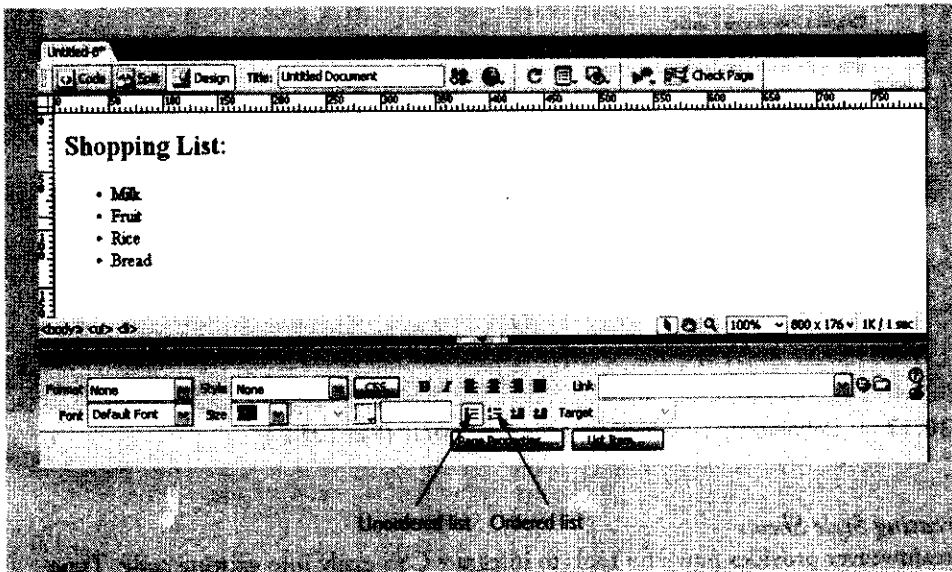


Fig. 20.11 | List creation in Dreamweaver.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml">
4 <head meta-equiv="Content-type" content="text/html; charset=utf-8" />
5 <title>Untitled Document</title>
6 </head>
7 <body>
8 <h2>Shopping List</h2>
9 <ul style="list-style-type: none;">
10 <li>• Milk</li>
11 <li>• Fruit</li>
12 <li>• Rice</li>
13 <li>• Bread</li>
14 </ul>

```

Fig. 20.12 | Definition list inserted using the **Text** menu. (Part 1 of 2.)

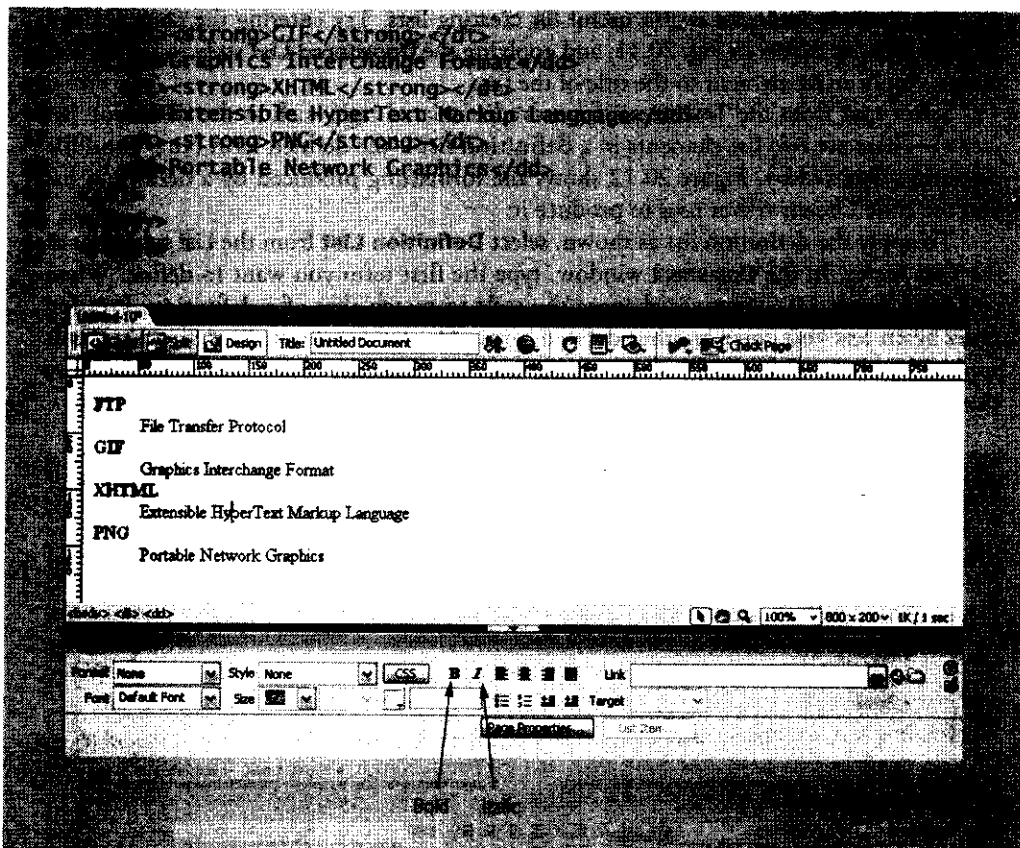


Fig. 20.12 | Definition list inserted using the **Text** menu. (Part 2 of 2.)

Creating Style Sheets

Dreamweaver provides powerful tools to integrate CSS easily into existing code. Type

```
Deitel Textbooks
Internet & World Wide Web How to Program, 4/e
Java How to Program, 7/e
Visual Basic 2005 How to Program, 3/e
C# For Programmers, 2/e
```

into the WYSIWYG display. Make the last four lines into unordered list elements using the method described above.

Select **CSS Styles** from the **Window** menu, or press **<Shift>-F11**. The **CSS Styles** panel will appear on the right-hand side of the page (Fig. 20.13). Now, click the **New CSS Rule** icon (Fig. 20.13) to open the **New CSS Rule** dialog. Next to the **Selector Type:** prompt, select the **Tag** option. This designates your style selections to the particular tag selected in the **Tag:** prompt. Enter **ul** into this menu's text box, or select it from the drop-down list. Next to the **Define in:** field, select the **This document only** radio button to create an embedded style sheet. The **(New Style Sheet File)** option generates an external style sheet.

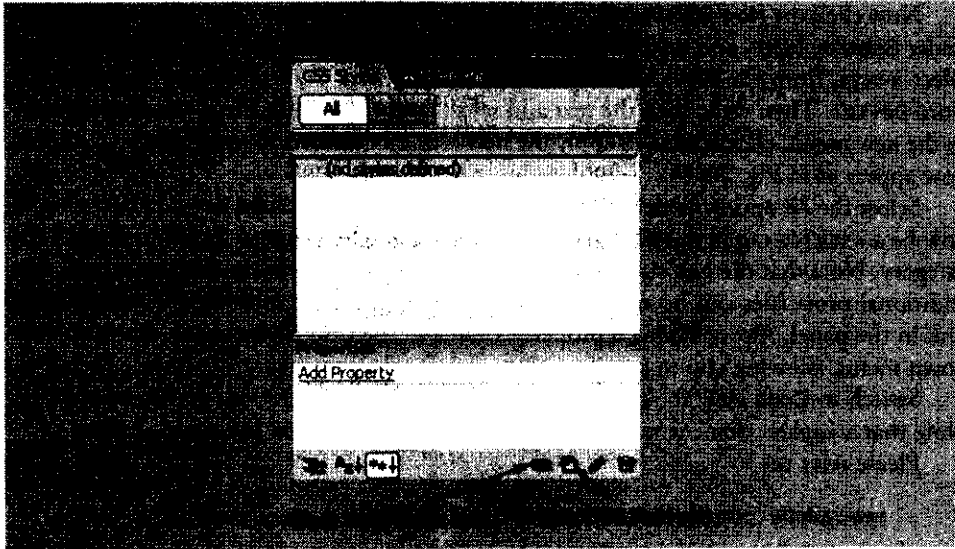


Fig. 20.13 | CSS Styles panel.

Click **OK** to open the CSS Rule definition dialog. **Type** should already be selected in the **Category** menu. Next to the **Decorations:** field, check the **underline** box. Now select **Background** from the **Category list**, and enter **#66ffff** into the **Background color:** field. Click **OK** to exit the dialog and return to the **Design** view. The text within the `<u>` and `</u>` tags should now appear as in Fig. 20.14.

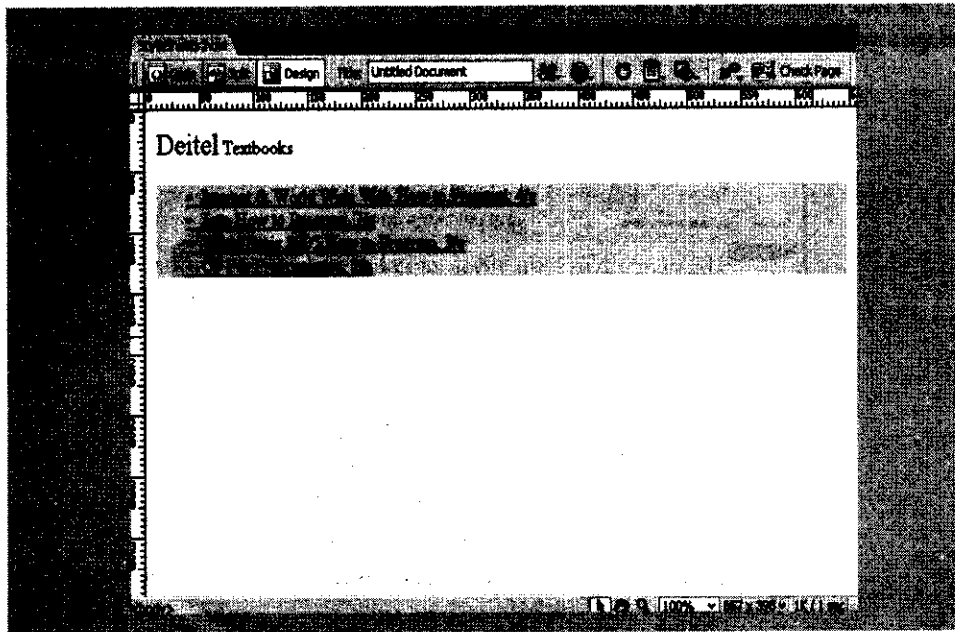


Fig. 20.14 | List with styles applied using CSS.

Now click the **New CSS Rule** icon to bring up the dialog again. This time, select **Class** under **Selector Type:**, and next to **Name:** enter `special`. In the **CSS Rule definition** dialog, select **x-large** from the **Size:** menu. Click **OK** to return to **Style** view, and highlight the word `Deitel`. Then right click the text and select **CSS styles** from the menu that appears. In the new menu, click **special** to apply the new class to the selected text. Your page should now appear as in Fig. 20.14.

Select the **All** option under the **CSS Styles** tab of the **CSS Styles** panel. There should now be a `<style>` tag in the **All Rules** window. Click the plus sign to its left to expand the category. Note that the two style rules that you created are present in this menu, and that additional properties can be added by selecting the rule, then clicking the **Add Property** link in the panel. Also, clicking a property's value in the **CSS Styles** panel creates a drop-down menu, allowing you to specify a new value for the property.

Switch to **Code** view to see the style sheet that Dreamweaver has generated for you. Note that a `` element was automatically created to contain the `special` class.

Please refer to

www.adobe.com/devnet/dreamweaver/css.html

for additional information on using CSS in Dreamweaver.

20.4 Images and Links

Inserting images using Dreamweaver is simply a matter of clicking a button and selecting an image to insert. Open the **Select Image Source** dialog (Fig. 20.15) either by selecting **Image** from the **Insert** menu, clicking the **Images** menu (Fig. 20.2) in the **Insert** bar and selecting **Image**, or pressing `<Ctrl><Alt>-I`. Browse your local hard-drive directory for a JPEG, GIF or PNG image. You can view the image's URL in the **URL** field—this will become the image's `src` attribute, which can also be viewed in the **Src** field of the **Property Inspector**.

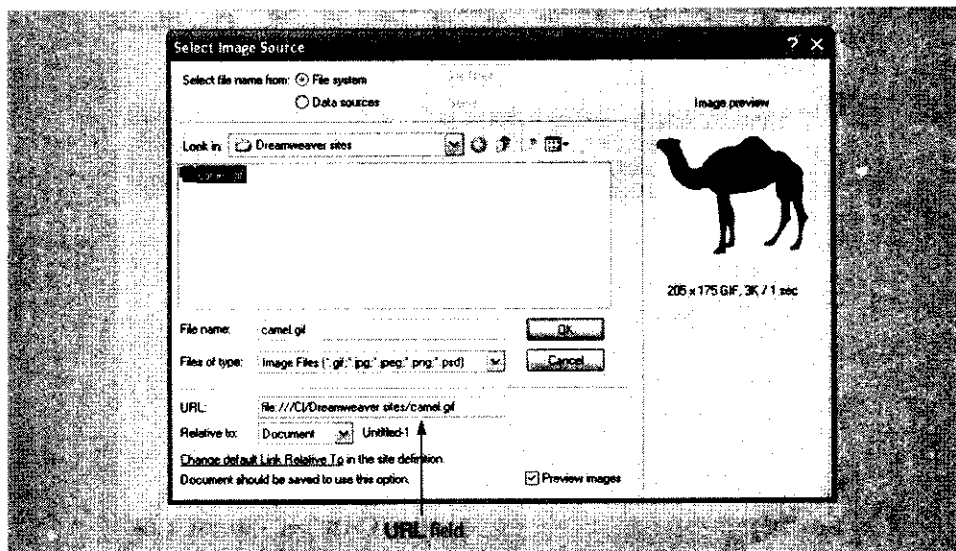


Fig. 20.15 | Image source selection in Dreamweaver.



Software Engineering Observation 20.2

When you insert a local image into an unsaved document, Dreamweaver sets an absolute path, such as `file:///C:/Dreamweaver sites/camel.gif`. If the image is stored in the same folder as the `.html` file, saving the document sets the image source to a relative path, starting at the folder in which the document is saved (e.g., `camel.gif`).

After inserting your image, select it in the **Document** window and create a hyperlink using the **Link** field in the **Property Inspector** (Fig. 20.16). Type in the URL to which the hyperlink will point, `http://www.deitel.com`. Using the **Border** field of the **Property Inspector**, add a `border = 0` attribute to the `` tag to remove the blue rectangle that normally appears around the image.

You can also change other image attributes in the **Property Inspector**. Try resizing the image using the height and width fields and changing its alignment in the **Align** pull-down menu. Clicking and dragging an image's borders also resizes the image.

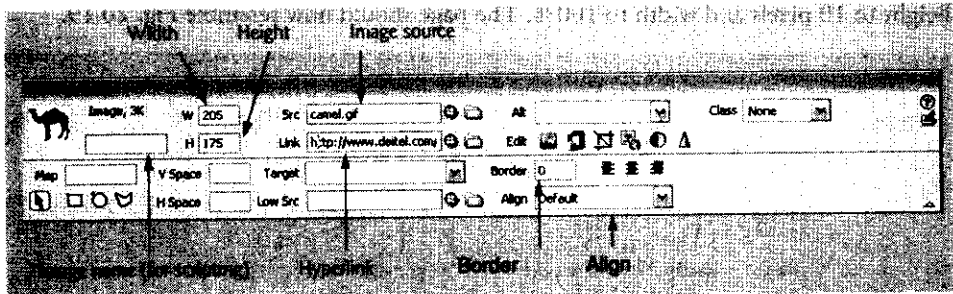


Fig. 20.16 | Image properties in the Property Inspector.

20.5 Symbols and Lines

Dreamweaver allows you to insert characters that are not located on the standard keyboard. These characters are accessed by selecting **HTML** in the **Insert** menu, then selecting **Special Characters**. Select **Other...** from the **Special Characters** submenu to view the **Insert Other Character** dialog, which contains a list of various characters (Fig. 20.17).

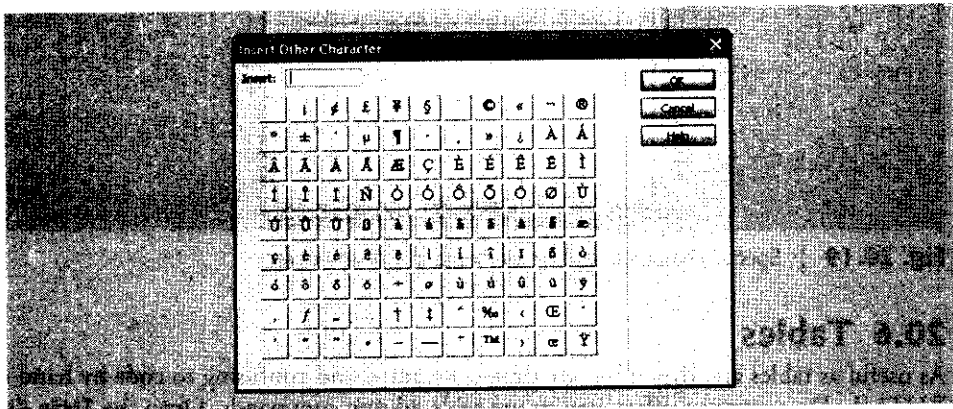


Fig. 20.17 | Insert Other Character dialog.

In the next example, we demonstrate how these symbols can be used in a web page, along with Dreamweaver's horizontal rule feature. Begin by typing

10 ÷ 5 =

Use the **Insert Other Character** dialog to insert the division symbol. Then, select **HTML** from the **Insert** menu and click the **Horizontal Rule** button. This action inserts a line (**hr** element) onto the page directly below the cursor's position. The line should be selected by default; if it is not, select the line by clicking it once. Using the **Property Inspector**, set the width to **60** pixels by entering **60** in the **W** field and selecting **pixels** from the pull-down menu directly to its right (Fig. 20.18). The other value in the menu, **%**, sets the line's length to the specified percentage of the screen. Make the line **5** pixels high by entering **5** in the **H** field (values in this field always have pixels as their units). Select **Left** from the **Align** pull-down menu.

On a new line, type the number 2. Insert another horizontal rule below the 2. Set its height to 10 pixels and width to 100%. The page should now resemble Fig. 20.19.

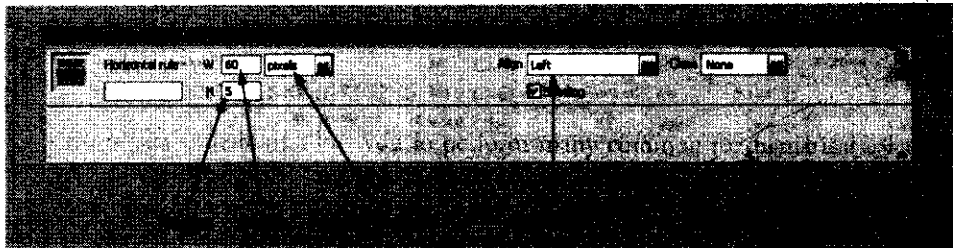


Fig. 20.18 | Horizontal Rule properties.

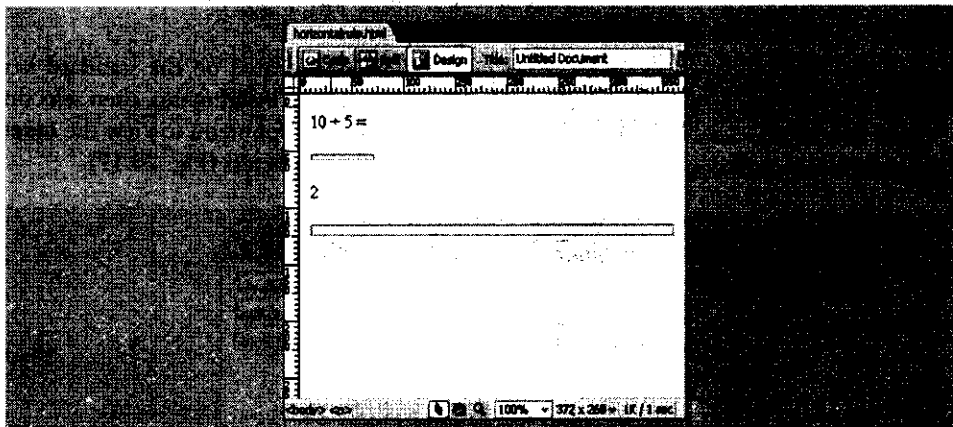


Fig. 20.19 | Special characters and **hr** elements in Dreamweaver.

20.6 Tables

As useful as tables are, they often are time consuming and confusing to code by hand in XHTML. Dreamweaver offers easy-to-use table-editing commands. Open the **Table** dialog by selecting **Table** from the **Insert** menu, clicking the **Table** button in the **Insert** bar or

pressing `<Ctrl><Alt>-T`. The **Table dialog** (Fig. 20.20) allows us to select the number of rows and columns, the width of the table and several other related settings.

Figure 20.21 is a simple table with two rows, two columns and no border. Once the table is placed, you can manipulate its size. Click in a cell and press `<tr>` in the **tag selector** (Fig. 20.2) at the bottom of the **Document** window to select that row. Pressing the **Delete** key removes the row from the table. You can highlight an individual cell by clicking `<td>` in the tag selector. Holding down the **Ctrl** key, then clicking multiple cells allows them all to be selected simultaneously. Clicking the **Merge Cells** button in the **Property Inspector** while two adjacent cells are selected combines the cells into one (Fig. 20.22). Dreamweaver uses the `colspan` and `rowspan` attributes of the `<td>` tag to merge cells. Select a cell and click the **Split Cell** button in the **Property Inspector** to open the **Split Cell dialog**, which allows you to divide the selected cell into any number of rows or columns (Fig. 20.23).

The **Property Inspector** allows us to manipulate the selected table, or a portion of the table. While a cell is selected, its text attributes can be adjusted just as we demonstrated earlier in the chapter. In addition, background and border colors can be assigned to cells, groups of cells or an entire table. We can adjust a cell's height and width in the **Property Inspector**. To manually adjust a cell's size, you can also click and drag its border lines.

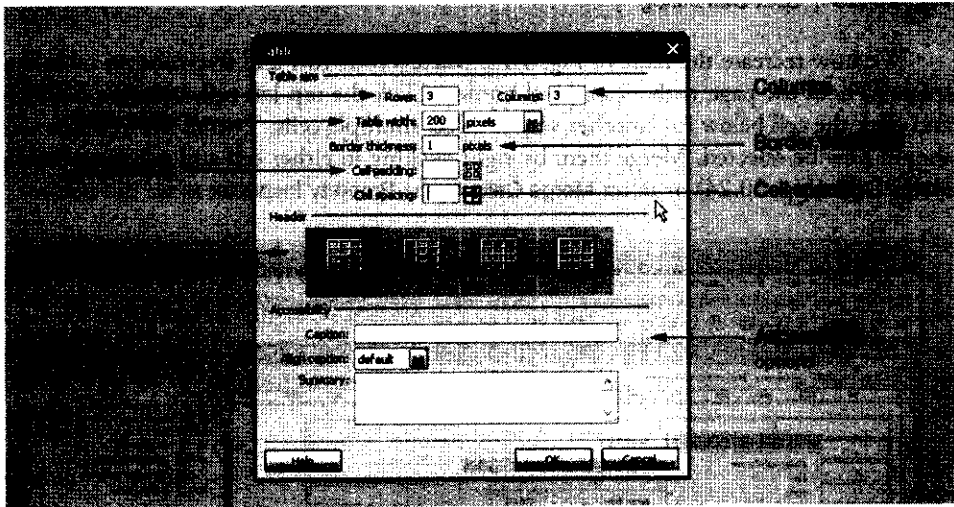


Fig. 20.20 | Table dialog.

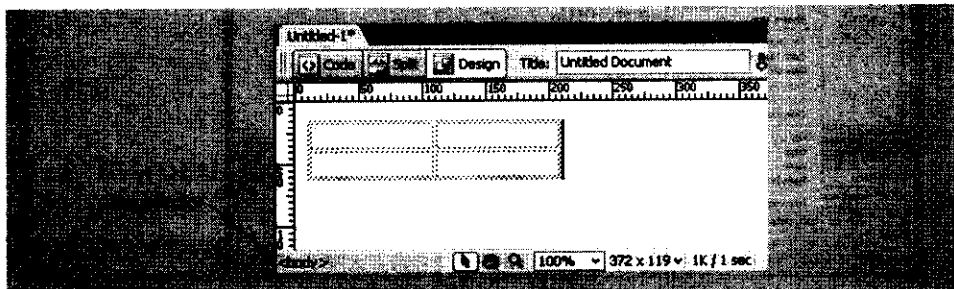


Fig. 20.21 | Table with two rows and two columns.

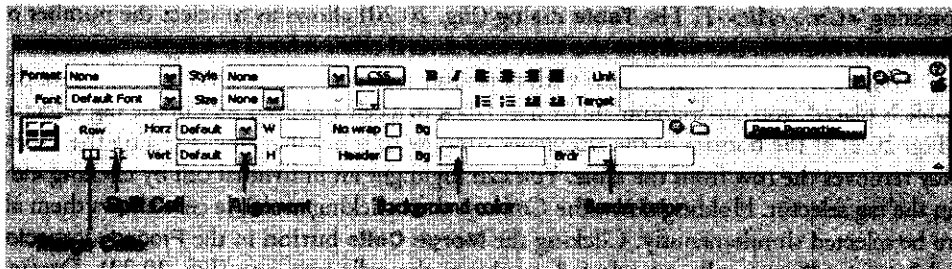


Fig. 20.22 | Table Property Inspector.

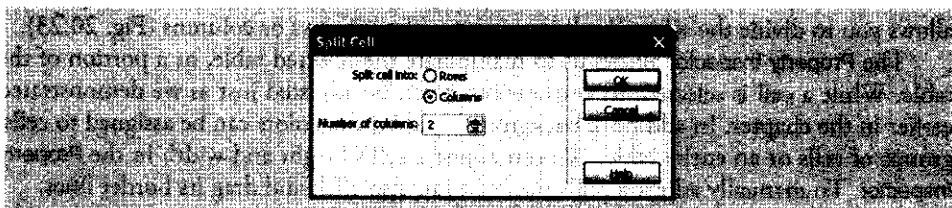


Fig. 20.23 | Split Cell dialog.

We now recreate the table of Fig. 4.11. Make a four-row and five-column table that spans 90% of the page with a one-pixel border. Click the top-left cell, hold the *Shift* key and click the cell below it—another way to select multiple cells. Two of the leftmost cells should now be selected. Merge them by right clicking in either cell and selecting **Table > Merge Cells** (Fig. 20.24) or select **Merge Cells** in the **Property Inspector** as we did before.

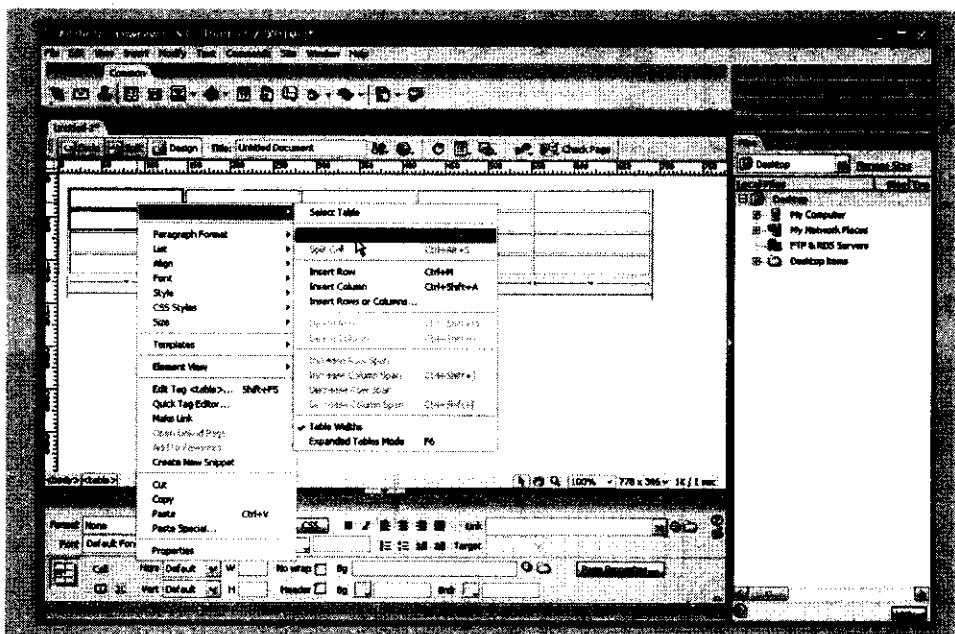


Fig. 20.24 | Merging cells in a table.

To make space for the title of the table, select the top four cells (again using the *Shift* key) and merge them together. The layout of the table should now resemble Fig. 20.25. Now, type in the text and insert the image.

To increase the visual appeal of the table, add color by selecting the desired cells and adjusting their background color in the **Property Inspector**. The size of rows and columns also can be adjusted by changing the **H** (height) and **W** (width) field values in the **Property Inspector** or by clicking and dragging the boundaries between cells.

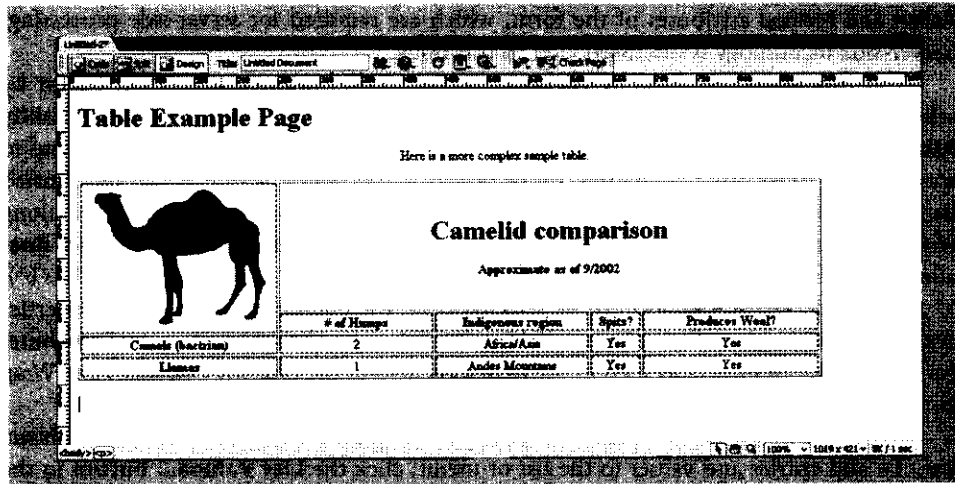


Fig. 20.25 | Almost completed table.

20.7 Forms

All the necessary XHTML coding needed for creating a feedback form or any other forms can be done in Dreamweaver. To insert a form, first select **Forms** from the insert menu in the **Insert** bar (Fig. 20.26). The **Insert** bar will now contain various options for creating forms. Click the leftmost button to insert an empty form into the document. Forms can also be inserted by selecting **Form** from the **Insert** menu's **Form** submenu.

After a form is inserted into a document, Dreamweaver displays a dotted line to delineate the bounds of the form. Any form objects (i.e., text fields, buttons, etc.) placed inside

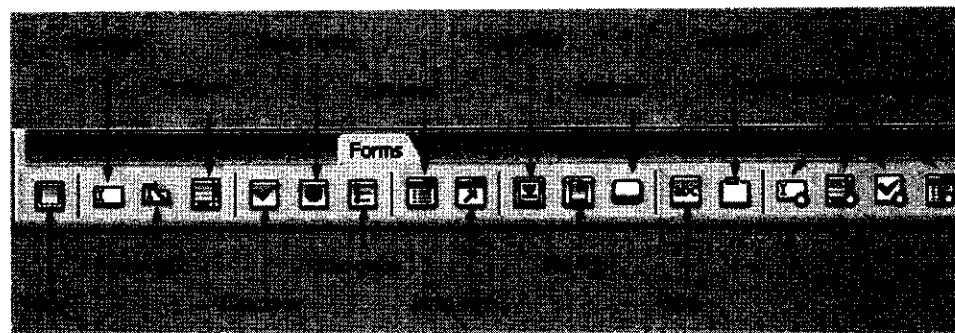


Fig. 20.26 | Forms Insert bar.

this dotted line will be part of the same form element in the XHTML code that Dreamweaver generates.

We can modify the properties of a form by clicking anywhere inside the dotted line that delineates the form, then clicking `<form#name>` (where *name* is the name of the form element) in the tag selector at the bottom of the Document window. Dreamweaver assigns default names to forms in sequential order (i.e., the first form inserted is named `form1`, the second form is named `form2` and so on). The name of the form can be altered in the **Form name** field in the **Property Inspector**. The **Property Inspector** can also be used to set the **Action** and **Method** attributes of the form, which are required for server-side processing. Server-side technology is discussed later in this book.

You can insert text field by clicking the **Text Field** button in the **Insert** bar or by selecting **Text Field** from the **Insert** menu's **Form** submenu. The **Input Tag Accessibility Attributes** dialog that appears allows you to set an `id` and `label` for the text field, and to specify some of its other properties. Once placed, a text field's attributes can be adjusted using the **Property Inspector**. Its name, `id` and `value` attributes can be set or modified along with the `size` and `maxLength` (Fig. 20.27). The text field type also can be set to **Multi line**, allowing multiple lines of text, or **Password**, making all entered text appear as asterisks (*).

Scrollable **Textareas** also can be selected from the **Form Insert** bar. Their properties are almost identical to those of a text field, except that they have the additional attributes for the number of lines (specified in the **Num lines** field in the **Property Inspector**) and **Wrap** (i.e., how the text area handles lines of text that exceed its width).

A drop-down select menu can be added by clicking the **List/Menu** button in the **Insert** bar. To add entries and values to the list or menu, click the **List Values...** button in the **Property Inspector** (Fig. 20.28). In the **List Values** dialog, you can add entries by pressing the + button, and remove entries by pressing the - button. Each entry has an **Item Label** and a **Value**. An entry can be made the default selection by selecting it in the **Initially selected** list in the **Property Inspector**.

Now that we've discussed the basics of forms in Dreamweaver, we're ready to create a "rate my website" form. To start, insert a form into a new page, followed by text fields, menus and text. The elements should appear as in Fig. 20.29.

Make the text fields the proper width by adjusting the **Char width** value in the **Property Inspector**. Now select the drop-down menu to the right of the text **How would you rate our site?** and click the **List Values...** button in the **Property Inspector** to add appropriate entries to the list (e.g., **Excellent**, **Good**, **Fair**, **Poor** and **Terrible**).

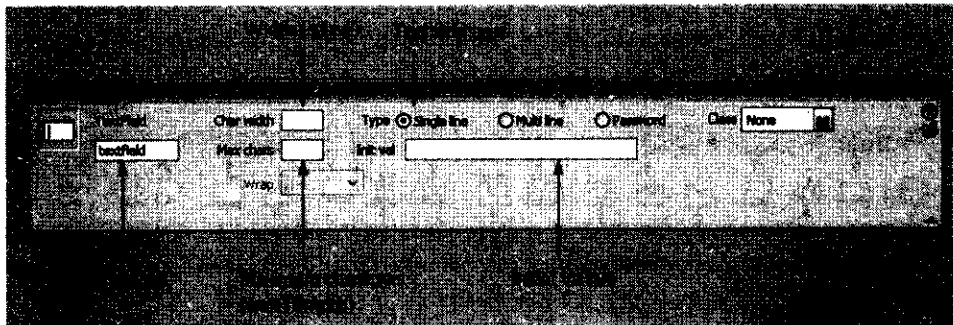


Fig. 20.27 | Text field Property Inspector.

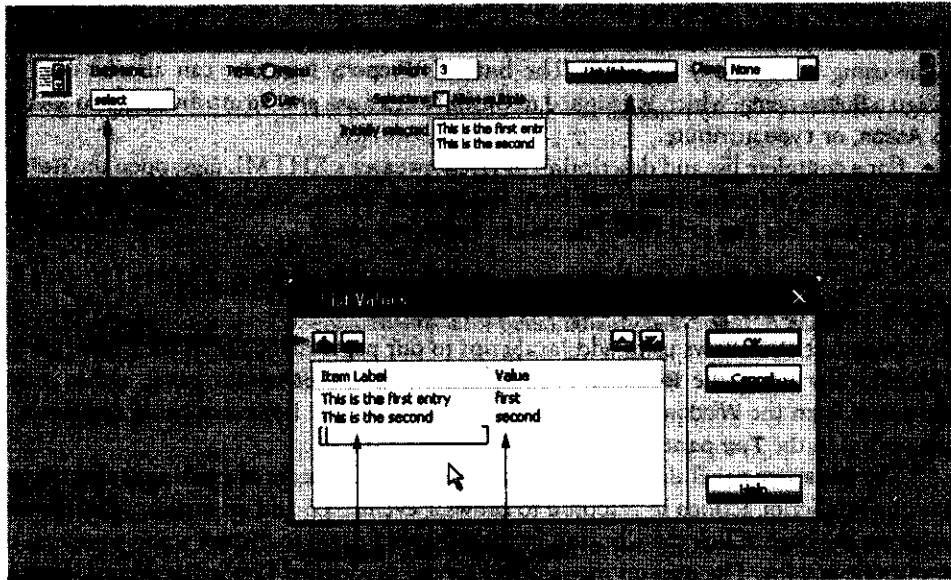


Fig. 20.28 | List Values dialog box.

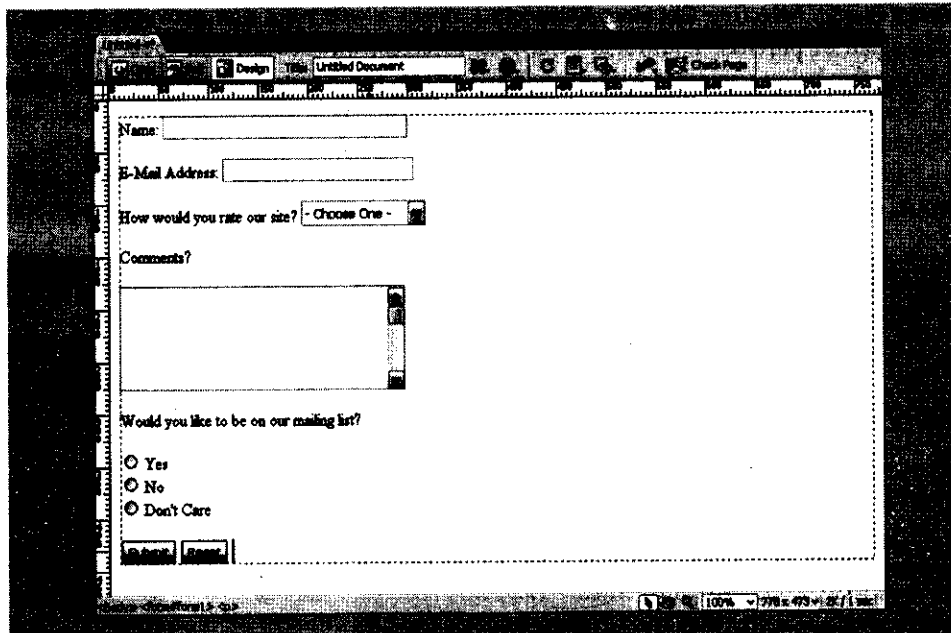


Fig. 20.29 | Completed form.

This example has three radio buttons, all contained in the same group. To add a group of radio buttons, click the **Radio Group** button in the **Insert** bar. In the **Radio Group** dialog, specify the **Name** of the group, and each radio button's **Label** and **Value**. The **Radio Group** dialog works similarly to the **List Values** dialog.

To create the **Reset** and **Submit** buttons, click the **Button** selection in the **Insert** bar. The **Value** of each new button defaults to **Submit**, but can be changed to **Reset** or any other value using the **Property Inspector**. The button's **Property Inspector** can also be used to assign a **Button name**, which is assigned to the button's name and id attributes, or to specify its **Action**, or type attribute.

For a complete list and description of Dreamweaver's XHTML tags, open the **Reference** panel by selecting **Reference** from the **Window** menu. Select the desired XHTML element from the **Tag** pull-down list in the **Reference** panel.

20.8 Scripting in Dreamweaver

Dreamweaver also allows us to add JavaScript to our pages manually in the **Code** view or automatically using the **Behaviors** panel. To open the **Behaviors** panel, either select **Behaviors** from the **Window** menu, or press <Shift>-F4. The **Behaviors** panel appears as a tab option in the **Tag** panel (Fig. 20.30).

The **Behaviors** panel allows us to add commands to elements of a web page that trigger various JavaScript actions in response to browser events. To add an action, select an element on the page. Click the + button in the **Behaviors** panel to display a pop-up menu of applicable actions. The pop-up menu offers several predefined JavaScript actions, such as **Go To URL** or **Popup Message**. A developer also can manually write an action by selecting **Call JavaScript** from the pop-up menu and entering the desired code into the **Call JavaScript** dialog. Selecting **Get More Behaviors...** opens a web page that provides options to download or purchase additional behaviors, extensions, functions and code. After completing the dialog associated with the selected action, the action and a default event appear in the **Behaviors** panel. A developer can change the event that triggers this action by clicking the event field and choosing an event from the drop-down list that appears.

Dreamweaver supports several server-side scripting languages, discussed later in the book, such as ASP.NET, JSF, PHP and ColdFusion. Server-side scripting elements, such

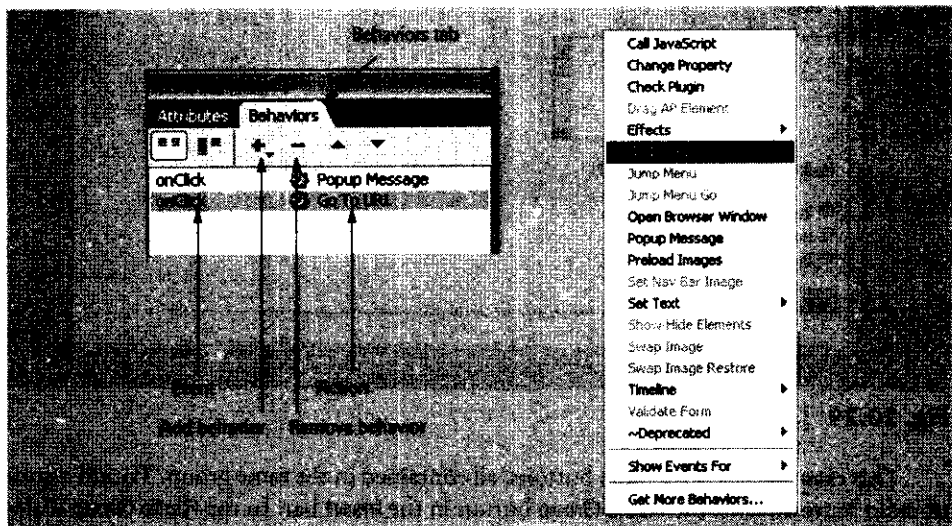


Fig. 20.30 | Behaviors panel and menu to add behaviors.

as **Databases** and **Bindings**, can be accessed in the **Window** menu. Tags of the various languages can also be selected from the **Tag Chooser**, which is accessed by selecting **Tag...** from the **Insert** menu, or from the icon in the **Insert** bar (Fig. 20.2). Dreamweaver allows the user to add scripting elements only where applicable.

20.9 Spry Framework for Creating Ajax Applications

Many toolkits are available that provide prebuilt controls to enhance web applications and make it easier to include JavaScript functions in your applications with minimal coding (such as the Dojo toolkit mentioned in Chapter 15, and the Prototype and Script.aculo.us toolkits in Chapter 24). Adobe also created its own toolkit for Dreamweaver to develop dynamic and more robust web pages known as the **Spry Framework**.

The Spry Framework enables web developers with basic knowledge of HTML, CSS and JavaScript to create richer websites and dynamic pages. The framework includes a ready-to-use JavaScript library, which contains prebuilt, but customizable, widgets (such as a **Validation Textarea**, **Validation Text Field** and a **Menu Bar**), effects (such as grow, shrink, fade and highlight) and Ajax capabilities. To view all of the available spry tools, click the **Spry** tab in the **Insert** bar (Fig. 20.31).

Recall that Ajax applications separate client-side user interaction and server communication and run them in parallel, making the delays of server-side processing more transparent to the user. Consider the form example that you built in Fig. 20.29. None of the data entered into the form is transmitted to the server until the user clicks the **Submit** button. At that time, any errors in the form are sent back to the user for correction. With Ajax and the Spry framework, text field input is validated on the client side. When the page loads, the files that provide the validation are loaded directly into the page, so you can check for errors in any given field as soon as the user moves to the next field in the form.

Now, let's rebuild the form in Fig. 20.29 using Spry controls. First, insert **Spry Validation Text Fields** next to the **Name** and **E-mail Address** labels.

Select the blue **Spry** box connected to the text field you created next to the **E-mail Address** label. In the **Property Inspector**, set the **Type**: to **Email Address**. Make sure that the **Change** checkbox is selected. This means that a valid e-mail address must be in the field and if any changes are made to the address, the client will display a message prompting the user to make a change before continuing (Fig. 20.33).

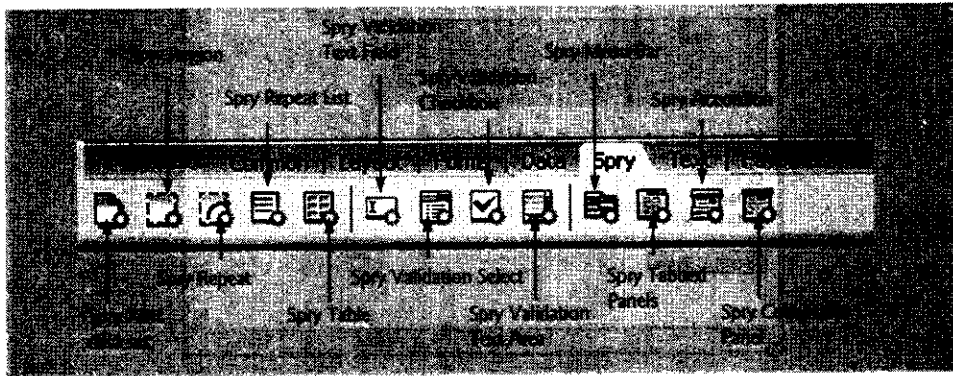


Fig. 20.31 | Spry Tools.

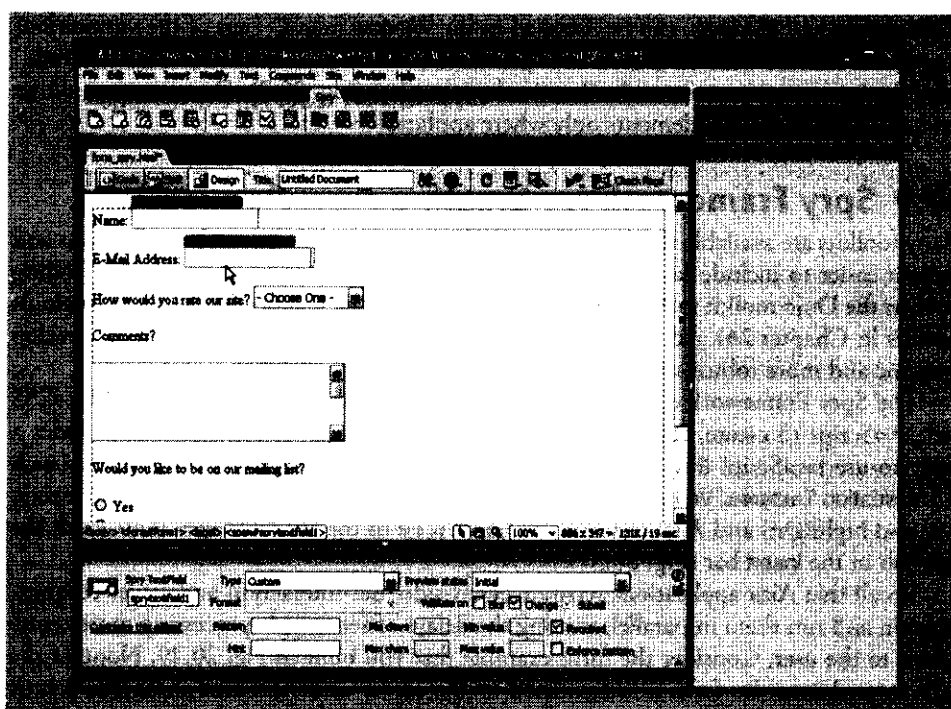


Fig. 20.32 | Inserting Spry Validation Text Fields.

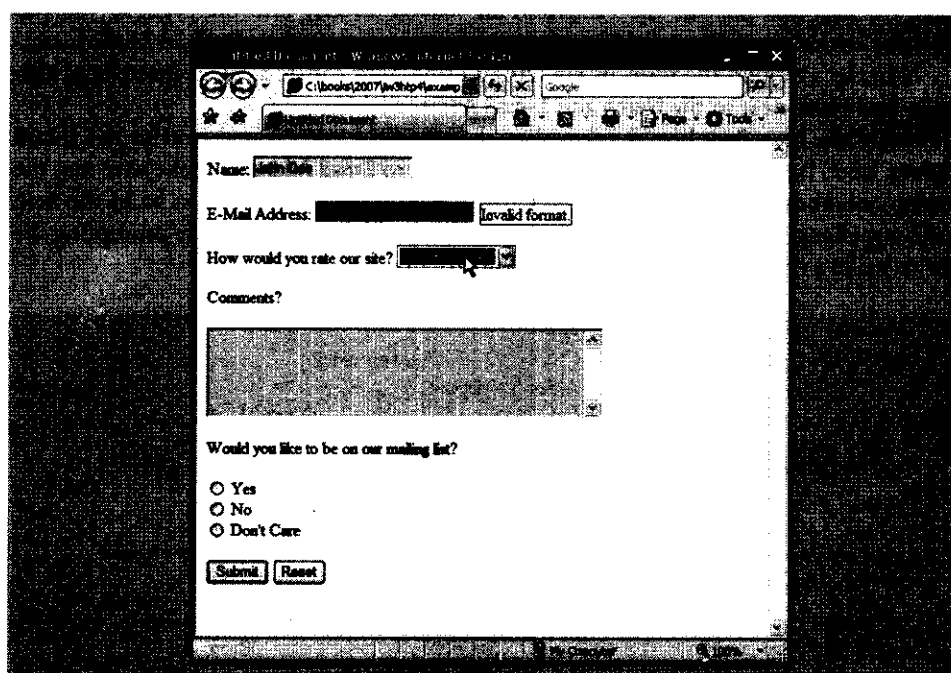


Fig. 20.33 | Using a Spry Text Field to validate data before continuing to fill out a form.

Our application detects an error when validating the information in the e-mail address text field. As soon as we try to move to the next field, the application displays the error **Invalid format** to let us know that we must correct the information that we originally typed into that field.

Real-time validation is a key element in Ajax and rich Internet applications. The framework also provides capabilities for loading and processing XML data obtained via Ajax interactions with the server. Using the Spry Framework, developers can take advantage of such rich functionality, even if they don't have a deep understanding of XML and JavaScript. For more information on the Spry framework, for Ajax-based examples and for the latest version of the framework, visit labs.adobe.com/technologies/spry/.

20.10 Site Management

In this book, we focus primarily on the skills and technologies involved in creating individual web pages. As a result, we do not spend much time discussing complete websites. Creating an effective website is a difficult process, requiring planning, effort and time.

Dreamweaver is a powerful tool for creating and maintaining a website. To create a site using Dreamweaver, first open the **Files** panel either by selecting **Files** from the **Window** menu or by pressing *F8*. Click the **Manage Sites...** link in the **Files** panel's drop-down list, or click the link to the right of this menu, to open the **Manage Sites** dialog. From this dialog, a developer can access previously created websites or create new ones. To create a new website, click the **New...** button in the **Manage Sites** dialog and select **Site** from the pop-up list. Then, follow the instructions provided by Dreamweaver's **Site Definition Wizard**. Once completed, site files can be viewed, accessed and added in the **File** panel.

In general, pages in a website should have consistent colors and styles to maintain site uniformity. Dreamweaver's **Assets** panel holds elements common to a website, such as pictures, colors and links. Open the **Assets** panel by selecting **Assets** from the **Window** menu or pressing *F11*.

While Dreamweaver is a valuable aid in website creation, it is not a replacement for thorough knowledge of XHTML and the related scripting languages taught in this book. Be sure to familiarize yourself with these other technologies before using Dreamweaver to accelerate the development process.

20.11 Web Resources

www.adobe.com/devnet/dreamweaver

Adobe's *Dreamweaver Developer Center* contains numerous tutorials and sample files intended for beginner, intermediate and expert users. This site explores some of the more advanced features of Dreamweaver in addition to the topics covered in this chapter.

www.adobe.com/software/dreamweaver

This site contains detailed product information, software downloads and links to featured sites created with Dreamweaver CS3.

Summary

Section 20.1 Introduction

- Dreamweaver CS3 is a popular HTML editor that can create complex XHTML elements, such as tables, forms and frames.

Section 20.2 Adobe Dreamweaver CS3

- Unlike editors that simply display XHTML code, Dreamweaver renders XHTML elements much as a browser would, using the WYSIWYG screen. This functionality enables you to design your web pages as they will appear on the web.
- Create a new document by selecting **New...** from the **File** menu. In the **New Document** dialog, select the **Blank page** tab from the leftmost column, and **HTML** from the **Page Type** list.
- Dreamweaver automatically encloses text in a paragraph (**p**) element for proper formatting.
- The **Category** list in the **Page Properties** dialog lets the user select a set of properties to view.
- To view or edit the XHTML that Dreamweaver has generated, you must switch from **Design** view to **Code** view.
- The tag names, attribute values and page text are all displayed in different colors.
- To save your file, click **Save** in the **File** menu or press **<Ctrl>-S**.
- To view your page in a browser, press **F12** or select **Preview in Browser** from the **File** menu. Note that the **File** menu option provides several browsers in which to view your code—more browsers can be added with the **Edit Browser List...** option.

Section 20.3 Text Styles

- In Dreamweaver, we can alter text properties with the **Text** menu or the **Property Inspector**.
- Dreamweaver is prone to produce somewhat inefficient code.
- Dreamweaver is capable of extensive text formatting, such as creating mathematical formulas.
- Many useful text-formatting options are located in the **Text** menu and can be applied to highlighted code.
- You can also access most of the elements in the **Text** menu by right clicking highlighted text.
- Dreamweaver automatically inserts a matching end tag in **Code** view.
- The **Property Inspector** can be used to create lists.
- Dreamweaver can integrate CSS easily into existing code using the **CSS Styles** panel. You can create both embedded and external style sheets with this tool.

Section 20.4 Images and Links

- Images can be inserted into Dreamweaver by selecting **Image** from the **Insert** menu or clicking the **Image** button in the **Insert** bar.

Section 20.5 Symbols and Lines

- Dreamweaver allows you to insert characters that are not located on the standard keyboard by selecting **HTML** in the **Insert** menu, then selecting **Special Characters**.
- Select **HTML** from the **Insert** menu and click the **Horizontal Rule** button to insert a horizontal rule.

Section 20.6 Tables

- Open the **Table** dialog by selecting **Table** from the **Insert** menu, clicking the **Table** button in the **Insert** bar or pressing **<Ctrl><Alt>-T**.
- The **Table** dialog allows us to select the number of rows and columns, the overall width of the table and several other related settings.
- The **Property Inspector** allows us to manipulate the selected table, or a portion of the table.

Section 20.7 Forms

- To insert a form, first select **Forms** from the **Insert** menu in the **Insert** bar, which will now contain various options for creating forms.

- Dreamweaver displays a dotted line to delineate the bounds of the form. Any form objects (i.e., text fields, buttons, etc.) placed inside this dotted line will be part of the same form element in the XHTML code that Dreamweaver generates.
- We can modify the properties of a form by clicking anywhere inside the dotted line that delineates the form, then clicking `<form#name>` (where *name* is the name of the form element) in the tag selector at the bottom of the Document window.
- For a complete list and description of Dreamweaver's XHTML tags, open the **Reference** panel by selecting **Reference** from the **Window** menu. Select the desired XHTML element from the Tag pull-down list in the **Reference** panel.

Section 20.8 Scripting in Dreamweaver

- Dreamweaver also allows us to add JavaScript to our pages manually in the **Code** view, or automatically using the **Behaviors** panel.
- The **Behaviors** panel allows us to add commands to elements of a web page that trigger various JavaScript actions in response to browser events.
- Dreamweaver supports several server-side scripting languages such as ASP.NET, JSP, PHP and ColdFusion. Server-side scripting elements, such as **Databases** and **Bindings**, can be accessed in the **Window** menu.

Section 20.9 Spry Framework for Creating Ajax Applications

- The Spry Framework promotes the creation of richer websites and dynamic pages by incorporating XML into documents for those web developers with basic knowledge of HTML, CSS and JavaScript.
- To view all of the available spry tools, click the **Spry** tab in the **Insert** bar.
- Ajax applications, including the Spry Framework, separate client-side user interaction and server communication, and run them in parallel, making the delays of server-side processing more transparent to the user.
- With Ajax and the Spry framework, text field input is validated on the client side. When the page loads, the files that provide the validation are loaded directly into the page, so you can check for errors in any given field as soon as the user moves to the next field in the form.
- You can manipulate the properties of Spry elements by selecting the blue Spry box connected to the element you created, then using the **Property Inspector**.

Section 20.10 Site Management

- Dreamweaver can help you create and maintain a website with the **Files** panel and the **Site Definition Wizard**.
- Dreamweaver's **Assets** panel holds elements common to a website, such as pictures, colors and links.

Terminology

Assets panel
Background Color
Behaviors panel
Button button
Category list
Code view
CSS Rule definition dialog
dd element (definition; `<dd>...</dd>`)

Design view
dl element (definition list; `<dl>...</dl>`)
Document toolbar
Document window
dt element (defined term; `<dt>...</dt>`)
Files panel
Font field in Property Inspector
Form button in Insert bar

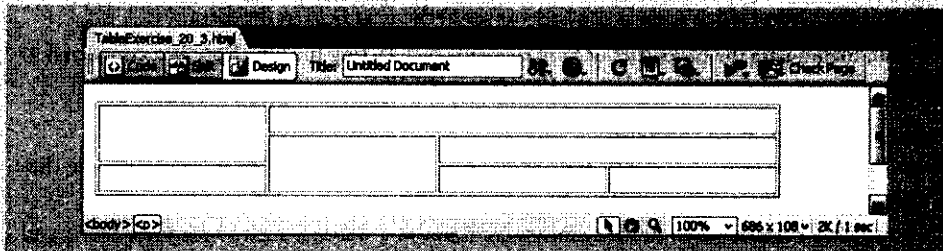
- Form tab in Insert bar
- Horizontal Rule in HTML option in Insert menu
- Images button in Insert bar
- Insert bar
- Insert menu
- Link field in the Property Inspector
- List Values button
- List/Menu button
- Manage Sites dialog
- Merge Cells button in Property Inspector
- New CSS Rule dialog
- Page Property... dialog
- Property Inspector
- Preview in Browser
- Save in File menu
- Special Characters dialog
- Split Cell button in Property Inspector
- Style in Text menu
- Table button in Insert bar
- Table dialog
- Tag selector
- Text Field button
- Text menu
- WYSIWYG (What You See Is What You Get)

Self-Review Exercises

- 20.1** State whether each of the following is *true* or *false*. If *false*, explain why.
- a) Dreamweaver renders XHTML elements correctly in its WYSIWYG display.
 - b) Dreamweaver allows web-page authors to insert images simply by clicking a button and selecting an image to insert.
 - c) Dreamweaver requires the user to manually write special characters into the code.
 - d) Dreamweaver delineates a form element in the WYSIWYG editor with a dotted line.
 - e) Dreamweaver can be used to create only XHTML documents.
- 20.2** Fill in the blanks for each of the following statements.
- a) A(n) _____ editor renders web-page elements exactly as a browser would.
 - b) The _____ allows you to adjust the selected element's attributes.
 - c) Dreamweaver's _____ option combines selected table cells into one cell.
 - d) The _____ panel allows a developer to add JavaScript to an XHTML document.

Exercises

- 20.3** Create the following table using Dreamweaver:



- 20.4** Create a personal web page using Dreamweaver that features an image and a list of interests. Experiment with different text-formatting options. Link the image to your favorite website.
- 20.5** Recreate the page in Fig. 5.2 using an external style sheet.