Introducing Microsoft® Silverlight™ 2, Second Edition

Laurence Moroney

To learn more about this book, visit Microsoft Learning at
http://www.microsoft.com/MSPress/books/12086.aspx

9780735625280

Microsoft Press

Table of Contents

Part I Introducing Silverlight 2

Part II Programming Silverlight 2

Chapter 2

Using Expression Blend with Silverlight 2

Expression Blend is a professional design tool intended to create engaging experiences for Windows and the Web. It allows you to blend all the necessary design elements for your Web experiences, including video, vector art, text, animation, images, and other content such as controls, with one set of tools. Expression Blend is designed to aid you in the building of Windows-based as well as Web-based applications. This chapter will introduce you to this tool, giving you a tour of what is possible with it. Expression Blend has far too many aspects to cover in one chapter, but by the end of this chapter, you'll have a good grasp of the basics and will be ready to delve further into the features of this wonderful tool on your own!

Getting Started with Expression Blend

Expression Blend is available as part of the Microsoft Expression suite. Details are available at *http://www.microsoft.com/expression.*

After you've downloaded and installed Expression Blend, launch it from the Start menu. You'll see the Blend integrated development environment (IDE), as shown in Figure 2-1.

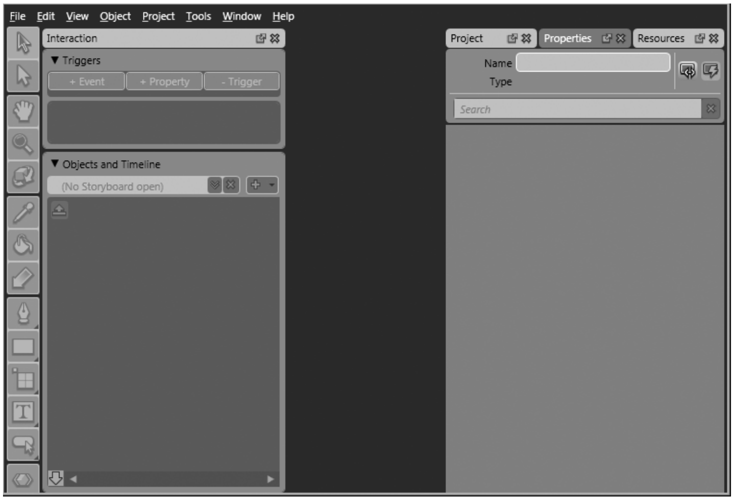
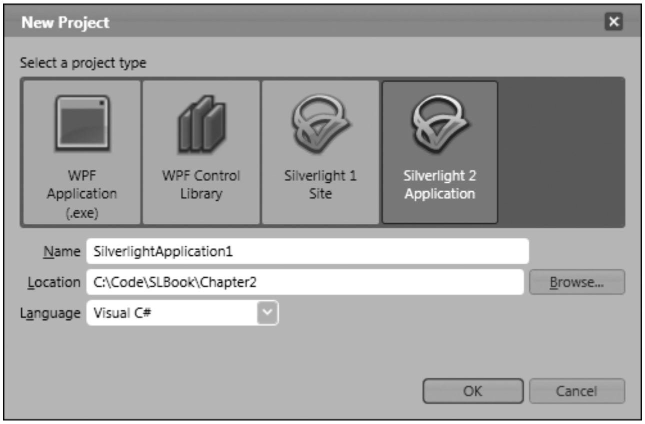


FIGURE 2-1 Expression Blend IDE.

To create a new application, select New Project from the File menu to open the New Project dialog box, as shown in Figure 2-2.



Name: SilverlightApplication1
Location: C:\Code\SLBook\Chapter2
Language: Visual C#

FIGURE 2-2 New Project dialog box options in Expression Blend.

The options that you are given are:

- **WPF Application (.exe)** This option creates a client-executable application built on the Windows Presentation Foundation (WPF); this type of project is a *Windows Only* application.
- **WPF Control Library** This option creates a DLL file that may be used for shared controls across WPF applications; this type of project is a *Windows Only* application.
- **Silverlight 1 Site** This option creates a Web site that uses the Silverlight control. It contains the basic JavaScript components to instantiate a Silverlight control as well as a sample XAML document with JavaScript-based event handlers, and it was covered in Chapter 1, "Introducing Silverlight 2." This option creates a *Web*-based and therefore *multiplatform* application.
- **Silverlight 2 Application** This option creates a Silverlight application based on the Silverlight 2 runtime. This application includes the .NET Framework runtime that supports your .NET-based applications, allowing them to run in a browser. When you select this type of project, you'll be able to pick your preferred programming language (either Microsoft Visual Basic or Microsoft Visual C#). This option also creates a *Web*-based and therefore *multiplatform* application.

Chapter 1 gave you some details about the Silverlight 1 Site template, and you can refer back to that chapter for more information about this option. For the rest of this chapter, we'll be focusing on the Silverlight 2 Application template.

Creating a Silverlight 2 Application

Open the New Project dialog box to create a Silverlight 2 application, and name your new project TestApp. Expression Blend will create a new project for you that contains everything you need for a Silverlight 2 .NET application.

You can see the project structure that it creates in Figure 2-3. This is identical to the product structure that is built by Visual Studio, which will be discussed in much more detail in Chapter 3, "Using Visual Studio with Silverlight 2."

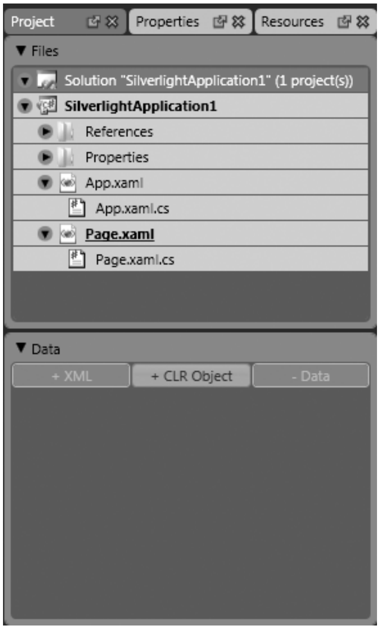


FIGURE 2-3 Silverlight 2 project structure.

What's important to note about the structure is that there are two XAML files in this application, and neither of them are a Silverlight XAML page, as in Silverlight 1. This is one way in which a Silverlight 2 Application project differs fundamentally from a Silverlight 1 Site project.

The Default *Page*

Silverlight 2 deals with your XAML as a *Page*. Thus, the template creates your default application XAML content as a file named Page.xaml. You'll see that the root of this, not surprisingly, is a *UserControl* and not a *Canvas*, as you may have been familiar with from Silverlight 1.

Following is the XAML for the default Page.xaml:

```
<UserControl
        xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d"
        x:Class="TestApp.UserControl1"
        d:DesignWidth="640" d:DesignHeight="480">

        <Grid x:Name="LayoutRoot" Background="White" />
</UserControl>
```

Note the use of *<UserControl>* to host the content.

You'll see a small difference between this XAML and that created by Microsoft Visual Studio (in Chapter 3). For example, this XAML has a few extra namespace declarations. Blend uses these for parsing the XAML to render in the designer. They don't affect your design beyond that, and you can safely ignore these.

You'll see that UserControl1 has a code-behind file that is generated for you. This will be named UserControl1.xaml.cs or UserControl1.xaml.vb, depending on which language you selected when creating the project. The file contains the basic code required to construct the *UserControl*. You can see it here:

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace TestApp
{
  public partial class UserControl1 : UserControl
  {
    public UserControl1()
    {
      // Required to initialize variables
      InitializeComponent();
    }
  }
}
```

You aren't *restricted* to using this file for your application logic. You can, of course, create other .cs (or .vb) files that can contain shared logic, but this one will be launched whenever the control is instantiated by the Silverlight runtime.

The Default App.xaml and Code-Behind Files

App.xaml and App.xaml.cs define the startup conditions for your application. These will be the first things loaded and executed by Silverlight on startup and the last things closed when the application is shut down.

This is accomplished using the *OnStartup* and *OnExit* events. These are set up for you by the project template. Note that UserControl1 does not render by default—it has to be instructed to render as part of the applications startup. This is accomplished in the *OnStartup* event handler, where the *RootVisual* for the application is set to an instance of *UserControl1:*

```
public App()
{
  this.Startup += this.OnStartup;
  this.Exit += this.OnExit;
  InitializeComponent();
}

private void OnStartup(object sender, StartupEventArgs e)
{
  // Load the main control here
  this.RootVisual = new Page();
}

private void OnExit(object sender, EventArgs e)
{
}
```

App.xaml does not support visual elements directly, so you cannot add controls or other visual elements directly. Just because it is XAML, don't think of it as a design surface. In this case, XAML is used for definition purposes only. For example, you can define application-specific resources for your application using it.

App.xaml.cs is useful for initialization of data that you want to use across several user controls. Keep this in mind as you design your application. For example, you could store some text that could be used across your application by declaring it as a resource in your App.xaml:

```
<Application
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="TestApp.App">
    <Application.Resources>
        <TextBlock x:Key="txtResource" Text="Hello"></TextBlock>
    </Application.Resources>
</Application>
```

You can now easily access this content from any control in your application as follows:

```
            TextBlock t = (TextBlock)Application.Current.Resources["txtResource"];
            string strTest = t.Text;
```

Executing the Application

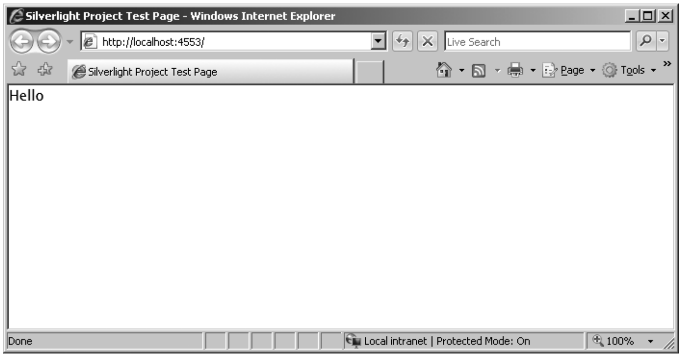
One thing that you may see is *missing* if you are sharp-eyed is a page to host the Silverlight control. Don't worry! Blend will automatically generate one for you. You'll see this when you launch the application.

Before going any further, add a simple *TextBlock* to your *UserControl* to render some text. Here's an example:

```
<UserControl
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d"
  x:Class="TestApp.UserControl1"
  d:DesignWidth="640" d:DesignHeight="480">

  <Grid x:Name="LayoutRoot" Background="White" >
    <TextBlock Text="Hello"/>
  </Grid>
</UserControl>
```

Now if you execute the application, you'll see something like the output shown in Figure 2-4.


Silverlight Project Test Page - Windows Internet Explorer
http://localhost:4553/
Live Search
Silverlight Project Test Page
Page
Tools
Done
Local intranet | Protected Mode: On
100%

Hello

FIGURE 2-4 Running the Silverlight application from Blend.

This simple application runs using the Cassini Web server (hence the random port number, 55924, that you can see in the address box in Figure 2-4) by generating an HTML page to host the Silverlight content.

Let's take a look at the source code for this page by using the browser command View Source. You can see the source code here:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
```

```
    <title>Silverlight Project Test Page </title>

    <style type="text/css">
    html, body {
     height: 100%;
     overflow: auto;
    }
    body {
     padding: 0;
     margin: 0;
    }
    #silverlightControlHost {
     height: 100%;
    }
    </style>

    <script type="text/javascript">
      function onSilverlightError(sender, args) {
        if (args.errorType == "InitializeError")  {
          var errorDiv = document.getElementById("errorLocation");
          if (errorDiv != null)
            errorDiv.innerHTML = args.errorType + "- " + args.errorMessage;
          }
        }
    </script>
</head>

<body>
    <div id='errorLocation' style="font-size: small;color: Gray;"></div>

    <div id="silverlightControlHost">
    <object data="data:application/x-silverlight,"
    type="application/x-silverlight-2-b1" width="100%" height="100%">
      <param name="source" value="TestApp.xap"/>
      <param name="onerror" value="onSilverlightError" />
      <param name="background" value="white" />

      <a href="…"
        style="text-decoration: none;">
      <img src="…"
        alt="Get Microsoft Silverlight" style="border-style: none"/>
      </a>
    </object>
    <iframe style='visibility:hidden;height:0;width:0;border:0px'></iframe>
    </div>
</body>
</html>
```

Do take note of the *<object>* tag. This attempts to instantiate Silverlight, and should it fail, it renders an image with a hypertext reference (HREF) to the Silverlight download in its place. You'll see more about this and other ways of instantiating the Silverlight object in Chapter 6, "The Silverlight Browser Control."

The Expression Blend IDE

Expression Blend offers a flexible IDE that is designed to maximize the amount of information on the screen while keeping it easy for the user to understand what is going on and not be overwhelmed.

The IDE has two main application workspace layouts: the Design workspace, which is used primarily for constructing and customizing your user interface (UI), and the Application workspace, which is used primarily for designing your timeline-based animations. You can switch between the workspaces using the F6 key or by selecting the workspace you want from the Active Workspace options on the Window menu.

The screen is divided into *panes* in the Expression Blend IDE, and each of the panes has a fixed purpose, as you'll discover when we tour them now.

The Tools Pane

The tools pane is on the far left side of the screen. It contains *tools*, such as Paint or Clip, that can be used to manipulate any object; *visual elements*, such as a *Rectangle* or *Ellipse*; *layout elements*, such as the *StackPanel* or *Canvas;* and *controls*, such as *Button* or *TextBox*). You can see the tools pane in Figure 2-5.

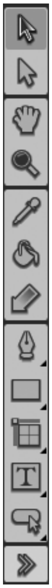


FIGURE 2-5 Expression Blend tools pane.

In Blend, similar tools can be collected together into a single icon on this tool pane. If you look at Figure 2-5, you can see how to view a set of similar tools by finding the white triangle in the lower-right corner of the tool. When this triangle is present, you can hold down the mouse button on that tool to find more members in the same "family" as the selected object. So, for example, if you hold down the mouse on the Rectangle tool, you'll see a pop-up box that shows you the other available shapes, as you can see in Figure 2-6.

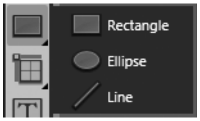


FIGURE 2-6 Grouped tools.

One nice shortcut that Blend provides is the way it creates a default tool on the toolbar when you have used a tool from the family of tools. That is, the tool that you just used will be displayed on the toolbar, so you don't need to hold down the mouse, wait for the menu, and then select the tool again to use it the next time.

So, for example, in Figure 2-6, the *Rectangle* is displayed on the toolbar, and when you hold down the mouse, you will see a box displaying the other visual element tools of this type that are available. If you then select the *Ellipse* and draw with it on the design surface, the toolbar will change to display the *Ellipse* instead of the *Rectangle*.

The Interaction Pane

The interaction pane, shown in Figure 2-7 and usually located just to the right of the tools pane, is designed to help you with the following tasks:

- View all of the objects on your design surface, including their hierarchy when you are using container objects.
- Select objects so you can modify them. This isn't always possible on the design surface because objects can be placed off screen or behind other objects.
- Create and modify animation timelines. You'll learn more about how to do this in the section titled "Using Blend to Design Animations" later in this chapter.

The interaction pane is designed to have two separate highlights. The currently selected object is highlighted in grey—in Figure 2-7, you can see that the *Border* control is highlighted in grey. This is the object that you can currently amend with the properties window or by dragging it around the design surface.

Although it appears grey in the figure, you can see on your screen that the *LayoutRoot* control has a yellow border around it. On the design surface, you'll also see this yellow border. This indicates that this is the currently selected container.

In addition to manipulating objects, you also use the interaction pane to create animations and storyboards. You do this by clicking on the plus sign (+) button at the top of the interaction pane. You'll explore the ways you can use this to create animations in the section titled "Using Blend to Design Animations" later in this chapter.

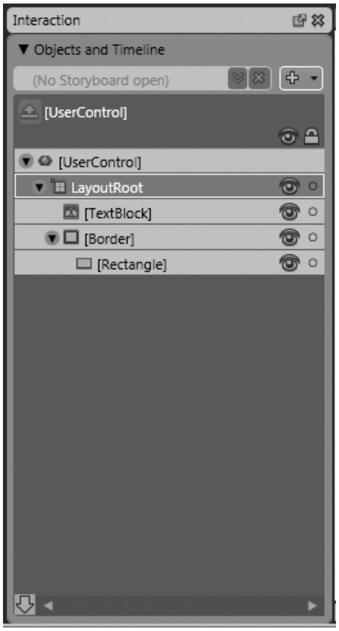


FIGURE 2-7 Interaction pane.

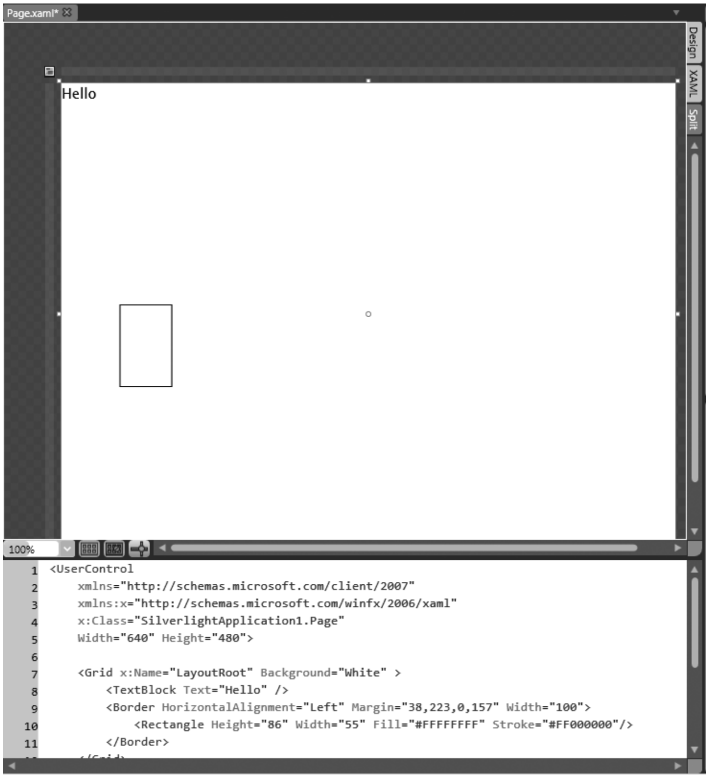
The Design Surface

The design surface is the main pane in the Expression Blend IDE screen, and this is where you can manipulate all the objects visually or by amending their underlying XAML code directly.

On the right side of the design pane, you will see three tabs:

- The Design tab gives you the pure design surface.
- The XAML tab gives you the XAML editing Window.
- The Split tab provides you with a split window—one half in design view and the other half in XAML view.

You can see the design pane in split view in Figure 2-8.



```
<UserControl
    xmlns="http://schemas.microsoft.com/client/2007"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SilverlightApplication1.Page"
    Width="640" Height="480">

    <Grid x:Name="LayoutRoot" Background="White" >
        <TextBlock Text="Hello" />
        <Border HorizontalAlignment="Left" Margin="38,223,0,157" Width="100">
            <Rectangle Height="86" Width="55" Fill="#FFFFFFFF" Stroke="#FF000000"/>
        </Border>
    </Grid>
```

FIGURE 2-8 Design pane in split view.

Note that you can use the Zoom feature in design view, so when you are working on sophisticated interfaces, you can zoom in for a detailed view and zoom out for an overview. You do this using the Zoom tool at the lower-left corner of the design pane. You can drop it down to select preset zoom settings, type the specific value you want in the box provided, or drag the mouse within it to set the desired zoom level.

The Project Pane

The project pane (shown in Figure 2-9) is used to manage the files in your project. The important thing to note in this pane is the use of context menus. Depending on *where* you right-click in this pane, you'll get a different (and appropriate) context menu. You might be familiar with context menus that provide commands for a specific pane, but in this case, you'll get different menus when you right-click the solution, the project, the References folder, and so on in the project pane.

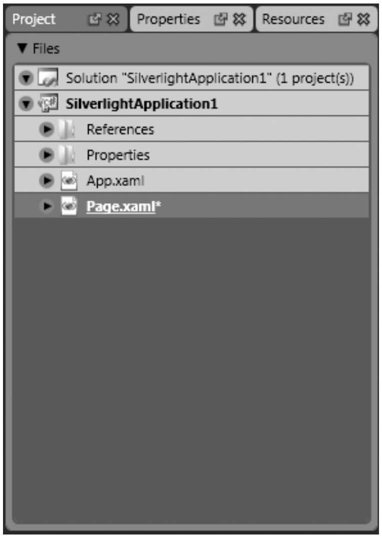


FIGURE 2-9 Project pane.

A *solution* is a collection of one or more projects. When you edit a solution, you can manage everything to do with the solution itself, including building, debugging, cleaning, and managing individual projects. In Figure 2-9, you can see the solution TestApp listed at the top of the project pane, and the pane indicates that there is one project within the solution.

A *project* is a collection of items that, when combined, make up an application that contains one or more Silverlight *Pages*. The project definition contains all the references to external components that this application needs within the References folder. When you right-click the project, the context menu that displays for the project allows you to manipulate the contents of the project, with options such as adding new items based on a template, adding existing items from other projects, or deleting items from the project.

The *References* folder within the project is used to manage references to precompiled assemblies that contain information that you want to use in your project. For example, if you want to use a custom control, it will be compiled into an assembly, so if you reference that assembly in your references, you can then use it within your application.

The *Properties* folder contains the application manifest file that describes all the properties of the project, including the list of references, so that the application can understand from where they are loaded at run time. The Properties folder should not be confused with the properties pane, indicated by the Properties tab at the top of the window shown in Figure 2-9 and explained in more detail in the following section.

The Properties Pane

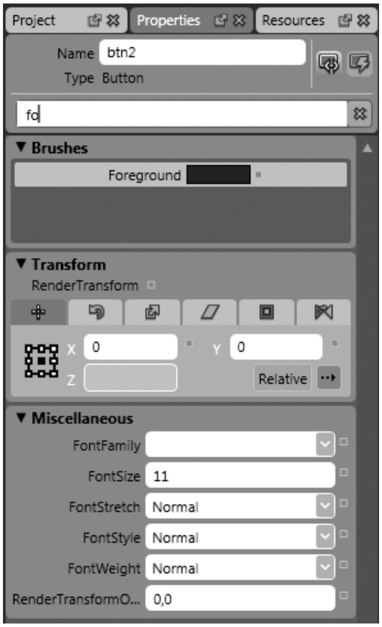
The properties pane is used to manage all the visual aspects of a particular element. Since XAML elements have many configurable properties, this pane gives you two very useful shortcuts.

The first shortcut is provided by the division of the properties pane into several classifications, typically providing access to the following visual aspects of elements:

- **Brushes** Allow you to set fill and stroke options as well as use an opacity mask on your element. You'll see a lot more detail about how brushes are used in Chapter 4, "XAML Basics."
- **Appearance** Allows you to set extended appearance properties for your object. Note that the available appearance properties will change drastically based on the object that you are currently editing. So, for example, if you are editing a *Rectangle* element, the Appearance section of the properties pane will allow you to set things like the corner radii, but if you are editing a *Button* element that doesn't have corner radii, you will not have this option available.
- **Layout** Allows you to edit the various layout options for your object, such as Width, Height, and Alignment options. You can also use layout options to change the position of an object within a grid—if the layout is on a grid.
- **Common Properties** Effectively the properties that are common across a *type* of object. So, for example, the common properties for controls that are distinct from shapes are typically edited here. These options can be very difficult to use, depending on the object that you are editing. For example, if you are editing a control, a common property will be its tab index, but if you are editing a shape, the tab index will not be available.
- **Transform** Provides you with the ability to edit the *RenderTransform* of your object. This defines how the object can be manipulated by the rendering system. Transformations are covered in detail in Chapter 5, "XAML: Transformation and Animation."
- **Miscellaneous** The catch-all location for properties that aren't available on any of the other classifications.

Do take note that these classification panes are further subdivided. You'll notice that many of them have an arrow at the bottom of the pane that can be used to expand and contract the properties view. This allows you to hide lesser-used properties until you need them.

The second shortcut in the properties pane is its Search feature, which allows you to search for a particular property. For example, if you know you want to edit some features of a font but don't know the name of the property itself, you can type **font** into the search engine, and the classifications and available properties will be filtered so that only those that have to do with fonts are displayed. This is done immediately upon a keystroke, so if you are searching for a font property—in our example, as soon as you type **fo**—you will see available properties displayed such as ***fo**reground* and *rendertrans**fo**rm* as well as the font properties, as shown in the list of properties displayed at the bottom of Figure 2-10.



Name btn2
Type Button
fo
Foreground
RenderTransform
X 0
Y 0
Z
FontFamily
FontSize 11
FontStretch Normal
FontStyle Normal
FontWeight Normal
RenderTransformO... 0,0

FIGURE 2-10 Using the properties pane.

Now let's take a look at how you can use all these tools we've introduced to build Silverlight applications.

Using Blend to Build Silverlight Applications

The main design-oriented functions that you can use Blend to accomplish as you put together your application include the following:

- Organizing the layout
- Placing and customizing visual elements
- Placing and customizing controls
- Designing animations

You'll explore each of these functions of Blend in the rest of this chapter.

Layout

In Silverlight, you use special tools to create and organize the layout of your application. There are several options available to you, and we will look at each of them in turn.

Using a Grid

The Grid layout element allows you to lay elements out in a structure that looks like a table. (Do not confuse the Grid layout element with a *Grid* control that gives you functionality similar to a spreadsheet.) When using a Grid layout tool, you can specify how your elements are placed by indicating their coordinates with virtual row and column designations within the Grid layout. For example, consider the following XAML:

```
<Grid x:Name="LayoutRoot" Background="White" >
  <Button Height="38" Margin="104,72,0,0" Width="58" Content="Button"/>
  <Button Height="24" Margin="210,72,0,0" Width="54" Content="Button"/>
  <Button Height="49" Margin="0,96,158,0" Width="80" Content="Button"/>
  <Button Height="54" Margin="297,185,270,0" Width="67" Content="Button"/>
  <Button Height="33" Margin="104,217,0,213" Width="87" Content="Button"/>
</Grid>
```

When rendered, this will appear as shown in Figure 2-11.

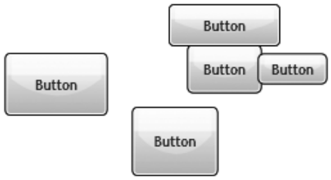
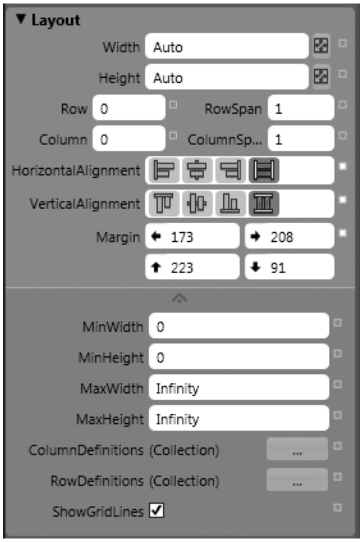


FIGURE 2-11 Random buttons.

Now, if you wanted to organize these buttons, you could carefully set their positions by dragging them around the design surface to place them at roughly the positions where you want them, but if you position them this way, you will need to zoom in to make sure pixels are aligned.

Alternatively, you could use the Grid layout, where you can use the layout properties of the button to determine its location in the grid. If you start with a new Silverlight project, you'll see that it has a Grid layout element on it called *LayoutRoot*. Select this element in your project, and look at the Layout properties associated with it. Expand the properties viewer until you see the settings for the ColumnDefinitions and RowDefinitions, as shown in Figure 2-12.



▼ Layout

Width Auto

Height Auto

Row 0 RowSpan 1

Column 0 ColumnSp... 1

HorizontalAlignment

VerticalAlignment

Margin 173 208

223 91

MinWidth 0

MinHeight 0

MaxWidth Infinity

MaxHeight Infinity

ColumnDefinitions (Collection) ...

RowDefinitions (Collection) ...

ShowGridLines ☑

FIGURE 2-12 Layout editor for a grid.

Because ColumnDefinitions and RowDefinitions are collections, each one has an ellipsis (...) button to the right of the setting name. This indicates that another dialog box will open when you click it. Select the button next to the ColumnDefinitions property setting, and the ColumnDefinition collection editor will display, as shown in Figure 2-13.

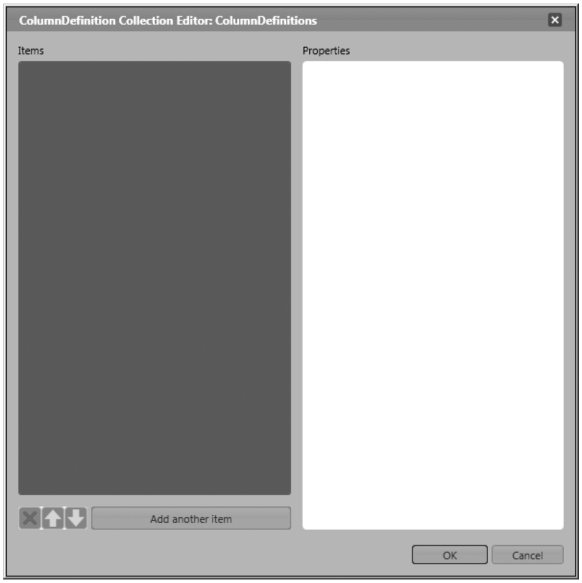


FIGURE 2-13 ColumnDefinition collection editor.

Use this dialog box to add, remove, and manage columns. Click the Add Another Item button three times to add three columns. Repeat this for the RowDefinitions property setting so that you have a grid that is comprised of three rows and three columns. After you have made these changes to ColumnDefinitions and RowDefinitions, you will see that the designer pane displays a 3 × 3 layout grid, as you can see in Figure 2-14.

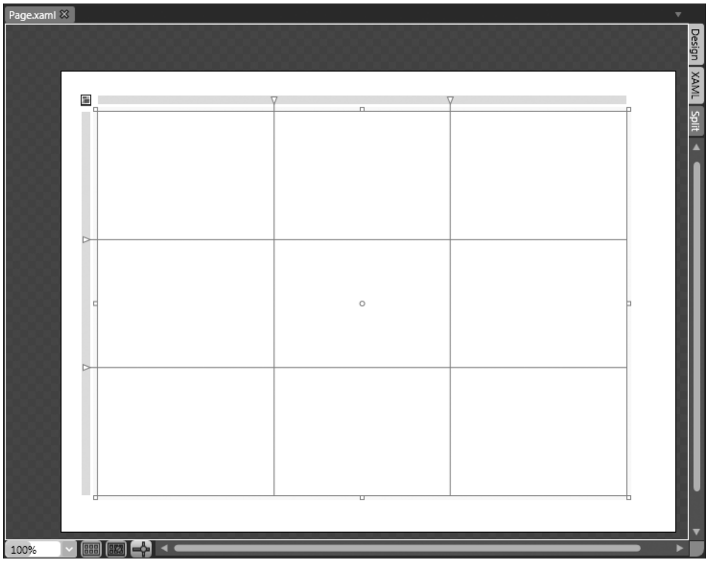


FIGURE 2-14 The 3 × 3 Layout grid.

Now, whenever you are placing an element on the screen, you'll see pink guidelines that show you how you can snap to a particular grid element, as shown in Figure 2-15. (They appear as wider grey lines in the figure.) Snapping the button to the grid and column layout like this will ensure that the button is always at that relative position and size in the grid.

Place another button in the central square on the grid, as shown in Figure 2-15. This time, do not snap it to the grid. Then run the application and experiment with resizing the window. You'll see that the first button will always remain at the same relative position and the same size, but the second button will change its width and/or height to stay relative to the size of the screen.

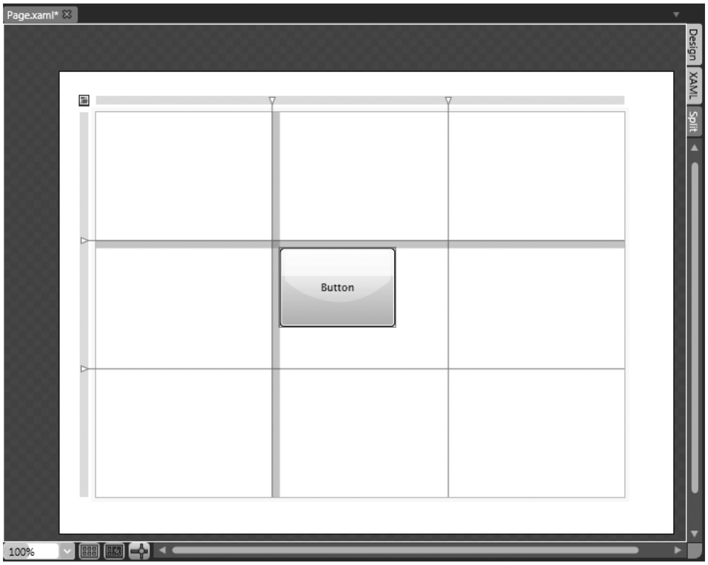


FIGURE 2-15 Using the Grid layout.

Using *Canvas*

The *Canvas* layout is a completely free-format drawing surface. You can specify the desired location for a control by setting its *Canvas.Top* and *Canvas.Left* properties or by using its *Margin* property.

So, for example, consider the following XAML:

```
<Canvas Height="261" Width="439">
  <Button Height="101" Width="110" Canvas.Left="101" Canvas.Top="82.5" Content="Button"/>
</Canvas>
```

You will see that the *Canvas.Top* and *Canvas.Left* properties for the button have been set. These indicate that the button will always be at those values *relative* to the parent *Canvas*, so as the *Canvas* moves, the button will move also. The *Canvas* layout is covered in more detail in Chapter 4.

Using *StackPanel*

The *StackPanel* layout will always orient its child controls either horizontally or vertically, stacking them (hence the name) based on the *Orientation* property. Note that the panel will override the positioning of the controls. For example, look at the following XAML:

```
<StackPanel Height="337" Width="224">
  <Button Canvas.Top="100" Height="64" Width="98"

     Orientation="Vertical" Content="Button"/>
  <Button Height="85" Width="92" Content="Button"/>
  <Button Height="48" Width="205" Content="Button"/>
</StackPanel>
```

You can see that the first button has its *Canvas.Top* property set to 100. You would expect that this would mean that the control would then be drawn at that position, but as Figure 2-16 shows, this is not the case, and it is stacked by the *StackPanel* layout at the top of the *StackPanel* (because the *StackPanel* has its *Orientation* property set to *Vertical)*.

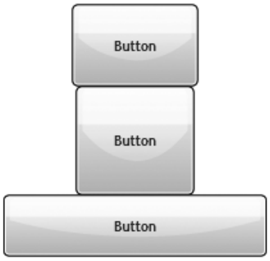


FIGURE 2-16 Buttons in a *StackPanel*.

When you have many controls in a *StackPanel*, you may go beyond the bounds of the Panel control itself, in which case the controls will be clipped to the bounds of the *StackPanel*. To get around this problem, you can use a ScrollViewer, which is explained in the next section.

Using the ScrollViewer

The ScrollViewer provides scroll bars that allow the user to pan around the contents of a layout if the contents exceed the bounds of the ScrollViewer. It can only contain one child control, so unless you are using a control that needs a large view area (such as an *Image*), it is typically only used to contain other containers.

For example, following is a *StackPanel* in which the contents exceed the vertical space available to it:

```
<StackPanel Height="300" Width="199">
  <Button Height="44" Width="86" Content="Button"/>
  <Button Height="57" Width="75" Content="Button"/>
  <Button Height="70" Width="59" Content="Button"/>
  <Button Height="109" Width="95" Content="Button"/>
  <Button Height="104" Width="88" Content="Button"/>
</StackPanel>
```

The *StackPanel* in this example is 300 pixels high, but the total height of all the buttons is 384 pixels, and so the bottom button will be cropped, as you can see in Figure 2-17.

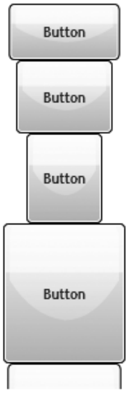


FIGURE 2-17 Cropped elements in a *StackPanel*.

Now, if you contain this within a ScrollViewer, you'll get better results. Note that the *StackPanel* will still crop the buttons if you do not change its height, so if you need to have an area of height 300, you can set the ScrollViewer to have this height and then set the *StackPanel* to have a different height. Here's the XAML to do this:

```
<ScrollViewer Height="300" Width="300">
  <StackPanel Height="400" Width="199">
    <Button Height="44" Width="86" Content="Button"/>
    <Button Height="57" Width="75" Content="Button"/>
    <Button Height="70" Width="59" Content="Button"/>
    <Button Height="109" Width="95" Content="Button"/>
    <Button Height="104" Width="88" Content="Button"/>
  </StackPanel>
</ScrollViewer>
```

You can see how the ScrollViewer created here appears in Figure 2-18.

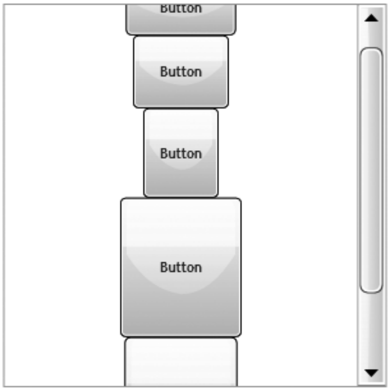


FIGURE 2-18 Using the ScrollViewer.

Now you can scroll up and down the button list, and the buttons will all be available; none are unavailable because of cropping if you use the ScrollViewer. Note that the button at the bottom in Figure 2-18 can be revealed by dragging the scroll bar down.

The Border Control

Not to be confused with the Border Patrol—part of the Department of Homeland Security—the Border Control is simply used to draw a border, background, or both around another element. For example, consider the following XAML:

```
<Border Height="318" Width="405" Background="#FFFF0000">
  <Button Height="234"
     HorizontalAlignment="Center"
     VerticalAlignment="Center"
     Width="239"
     RenderTransformOrigin="0.5,0.5"
     Content="Button">
  </Button>
</Border>
```

This will create a red background behind the button.

Placing and Customizing Visual Elements

The visual elements available are defined in the XAML specification, and you will learn about each of them in detail in Chapter 4. Right now, let's take a look at the basic shapes and tools that are available on the toolbar. These include the following shapes:

- ***Rectangle*** Select this shape to draw a straight-sided quadrilateral with 90-degree angles at each corner. You can make a square by creating a *Rectangle* with equal width and height properties.
- ***Ellipse*** Use this shape to draw an elliptical figure, an oval. You can make it a *Circle* by making the width and height properties equal.
- ***Line*** This shape simply draws a straight line between two end points.

There are also tools available on the toolbar that you can use to create free-form shapes:

- **Pen** Use this tool to draw a set of connected line segments represented by an underlying *Path* element.
- **Pencil** Use this tool to draw a set of connected elements, which can be lines or curves. Blend will take the strokes that the user draws and represent them with an underlying *Path* element.

Each of these visual elements, including those created with the Pen and Pencil tools, are represented by a single element, and this element can then be treated as any other object; that is, you can modify it in many ways, including setting its properties or animating it. For example, consider Figure 2-19, in which the Pencil tool has been used to draw a set of connected curves to create a representation of the word *Hello* in script. Look on the Objects And Timeline view, and you'll see the object represented as a *Path*.

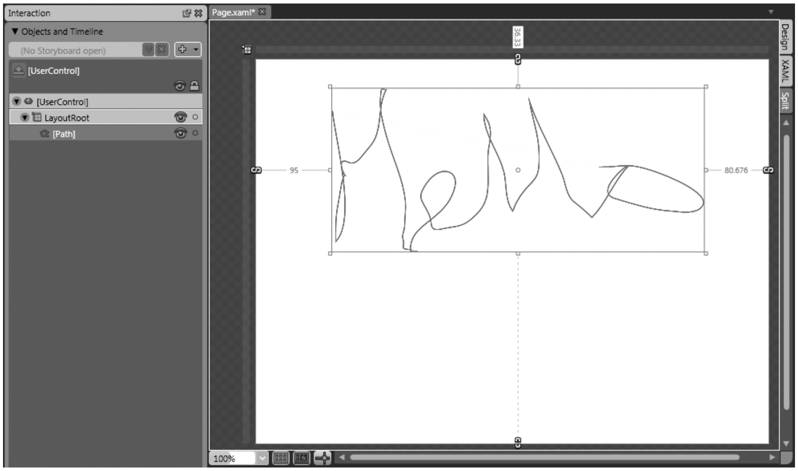


FIGURE 2-19 Editing a *Path* object.

Now, this pencil "drawing" of the word *Hello* is treated as a single object, so you can edit its properties, including *Fill*, *Brush*, and so forth, simply by selecting it from the Objects And Timeline view and then editing the properties in the property pane, as with any other object.

Placing and Customizing Controls

Controls are treated by Blend in exactly the same way as visual elements. You simply select them from the toolbar and draw them on the design surface. After you've created them on the design surface, then you can edit their properties. Controls are discussed in detail in Chapter 7, "Silverlight Controls: Presentation and Layout."

One thing to note is that Blend gives you two families of controls on the toolbar. The first includes the Text controls: *TextBlock* and *TextBox*. The second includes the set of basic user interface controls: *Button, CheckBox, ListBox, RadioButton, ScrollBar, Slider,* and *GridSplitter*.

Finally, the toolbar gives you the option to add controls that aren't part of this set. You can do this by selecting the Asset Library link at the bottom of the toolbar. This will display the Asset Library dialog box, as shown in Figure 2-20.

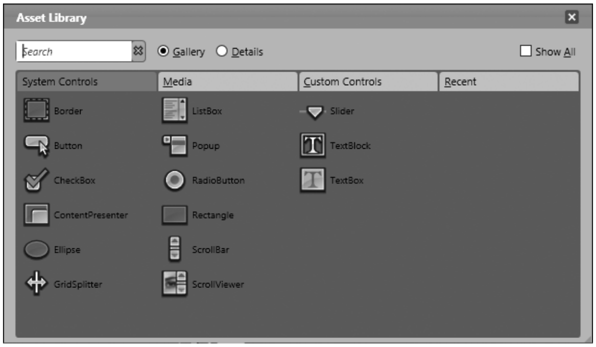


FIGURE 2-20 Asset Library dialog box.

You can select controls in the Asset Library dialog box to add them to the toolbar. You also can search for specific controls by entering the term in the search box. So, for example, if you want to use a *MediaElement* control, start typing the letters of the control's name. When you see the control you want (in this case, the *MediaElement*), you can select it, and it will then be available to you on the toolbar.

Then you can draw the control on the design surface and manipulate its properties with the properties editor, as you have done with the visual elements and layout controls.

Using Blend to Design Animations

We will examine how to create animations in detail in Chapter 5, but to put it succinctly, animations occur in Silverlight whenever a property of an object changes its value over time. You can design these kinds of animations visually in a very straightforward manner by using Blend and the timeline editor.

One form of animation that Silverlight supports is the *DoubleAnimation*, which is used to change numeric properties, such as the width of an *Ellipse* visual element. Another is the *ColorAnimation*, which is used to change the color of the *Brush* property.

For example, consider the *Ellipse* shown in Figure 2-21 (which appears as a circle because its height and width properties are equal). To visually design an animation that changes the width of this *Ellipse* element, you'll add a new *Storyboard* that contains the animation.

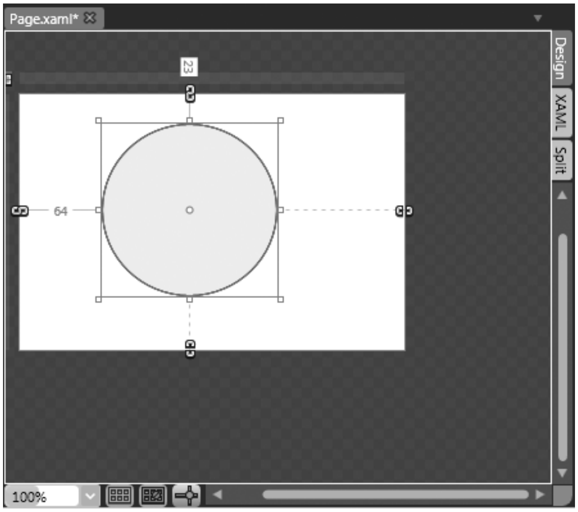


FIGURE 2-21 Drawing a circle.

On the Objects And Timeline view, select the *Ellipse* and then press the + button next to the *Storyboard* list at the top of the pane. Accept the defaults in the Create Storyboard dialog box that display, and then the Timeline editor will appear. You'll also see the message Timeline Recording Is On at the top of the Blend window. Press the F6 key to rearrange the workspace so that the timeline is displayed to make it easier to work on an animation. Your screen should look something like the one shown in Figure 2-22.

Look for the yellow line in the timeline view. This denotes the *current* position on the timeline. Drag it to the 2-second mark, and then click the Record Keyframe tool at the top of the timeline. It looks like a blob with a little green plus sign (+) at its lower-right side. You'll see a little oval that appears in the timeline at the 2-second mark, as shown in Figure 2-23.

Now that you have defined a keyframe, any changes that you make to the properties of the object will be recorded at that key frame, so go ahead and change the width of the *Ellipse* while the yellow line is still at the 2-second mark, indicating the current position of the timeline. For example, change the width to **200** and the *Fill* color to **Red**.

Now drag the playhead (the top of the vertical yellow line on the timeline) left and right, and you'll see a preview of the animation previewed, with the width and color of the circle shape changing over time.

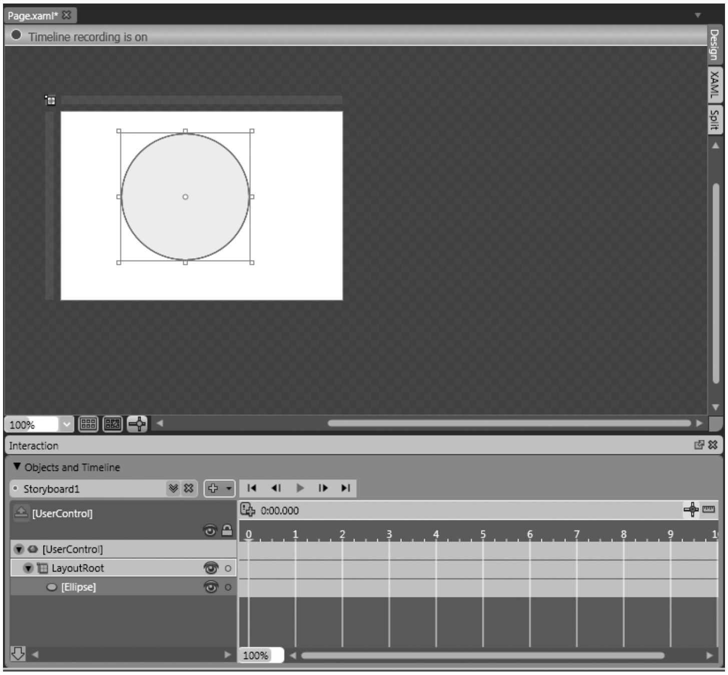


FIGURE 2-22 Editing the timeline.

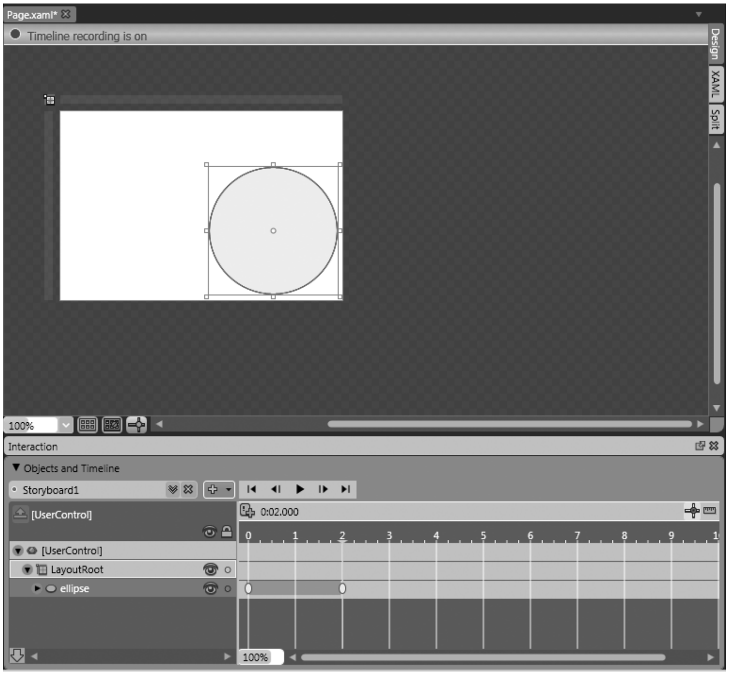


FIGURE 2-23 Adding a key frame.

You can see the XAML that is generated by your visual creation of the animation here:

```
<UserControl.Resources>
  <Storyboard x:Name="Storyboard1">
    <DoubleAnimationUsingKeyFrames Storyboard.TargetName="ellipse"
      Storyboard.TargetProperty="(FrameworkElement.Width)"
      BeginTime="00:00:00">
      <SplineDoubleKeyFrame KeyTime="00:00:02" Value="200"/>
    </DoubleAnimationUsingKeyFrames>
    <ColorAnimationUsingKeyFrames Storyboard.TargetName="ellipse"
      Storyboard.TargetProperty="(Shape.Fill).(SolidColorBrush.Color)"
      BeginTime="00:00:00">
      <SplineColorKeyFrame KeyTime="00:00:02" Value="#FFFF2200"/>
    </ColorAnimationUsingKeyFrames>
  </Storyboard>
</UserControl.Resources>
<Ellipse Height="100" Width="100" Fill="#FFFFF500" Stroke="#FF000000" x:Name="ellipse"/>
```

You'll delve into the structure of this XAML in much more detail in Chapter 5, but the important elements to note here are the *Storyboard.TargetName* instances that indicate which element the animation is being defined for, and the *Storyboard.TargetProperty* that indicates the property that is going to be changed. As you can see in this XAML, there are two animations, one that changes the width of the target and the other that changes its color. Silverlight then takes this definition and uses it to calculate the values required for each frame at the time the animation is rendered.

Summary

In this chapter, you learned the basics of working with Expression Blend, taking a quick tour of what it offers you as a designer or developer creating and implementing your own Silverlight applications. You saw how Blend can be used to create Silverlight solutions and projects, and then you saw what tools the Blend IDE offers you to add and manage visual elements, layout, controls, and animations in your application.

We've just begun to investigate what you can do with Blend in this chapter, but your introduction to Blend here may well inspire you to want to learn more about it.

The other half of the designer/developer workflow tool package is found in Visual Studio. In Chapter 3, you'll take a look at how you can use this tool, what it has in common with Blend, and what powerful features it provides for developers. You will have the chance to use Visual Studio to build your first Silverlight application—a sliding picture puzzle game.