C

Windows Presentation Foundation

WPF, previously known as Avalon, offers several features and functionalities to develop high-end Web applications. WPF supports various media, such as text, images, audio, and video, and allows you to work with two-dimensional (2-D) as well as three-dimensional (3-D) graphics. WPF is suitable for both User Interface (UI) designers and application developers. It provides a convenient platform for designers and developers to share their work and collaborate to accomplish their task. WPF also supports eXtensible Application Markup Language (XAML). XAML is a declarative markup language based on eXtensible Markup Language (XML) introduced by Microsoft Corporation. This markup language allows designers to easily and quickly define and describe the elements for the UI of WPF applications. Application developers can then use the UI elements defined using XAML to specify the application logic in the code-behind files in any of the .NET languages, such as C# or VB. However, you can also use the code-behind files to create UI elements at runtime. Consequently, WPF allows both segregation and integration of the UI aspect and application logic in WPF applications.

In this appendix, you learn about the architecture of WPF 4.0, features of WPF 4.0, and how to create XAML browser applications.

Let's start with a discussion on the architecture of WPF 4.0.

Architecture of WPF 4.0

The WPF architecture consists of several components, such as `PresentationFramework`, `PresentationCore`, `CLR`, `milcore`, `User32`, `DirectX`, and `Kernel`. Figure C.1 shows the architecture of WPF:

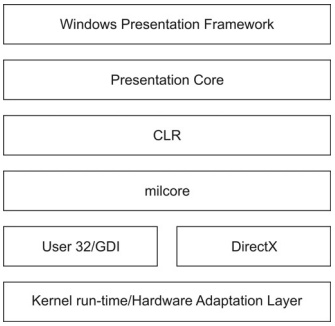

Figure C.1: Displaying the WPF Architecture

WPF 4.0 is a part of .NET Framework 4.0 and it consists of both managed and unmanaged components. The managed components of WPF 4.0 are based on Common Language Runtime (CLR) and make use of the different features of CLR, such as cross-language integration, type-safety and security, debugging and exception handling, and garbage collection. Most of the functionalities and features of WPF 4.0 are provided through the managed components. WPF 4.0 also contains an unmanaged component called the `Media Integration Layer` (`milcore`). This layer provides a complete replacement of `Win32` rendering and performs the task of drawing pixels. The `milcore` layer helps to keep Central Processing Unit (CPU) free for other processing by using computer's graphic card as much as possible. It is developed by using unmanaged code. Both `Presentation Framework` and `Presentation Core` (Figure C.1) layers are developed on the managed code. The `Presentation Core` layer contains base types that can be used to implement UI components. The `Presentation Framework` layer provides certain classes to implement user layouts and presentation features. Figure C.2 shows the hierarchy of core classes of WPF:

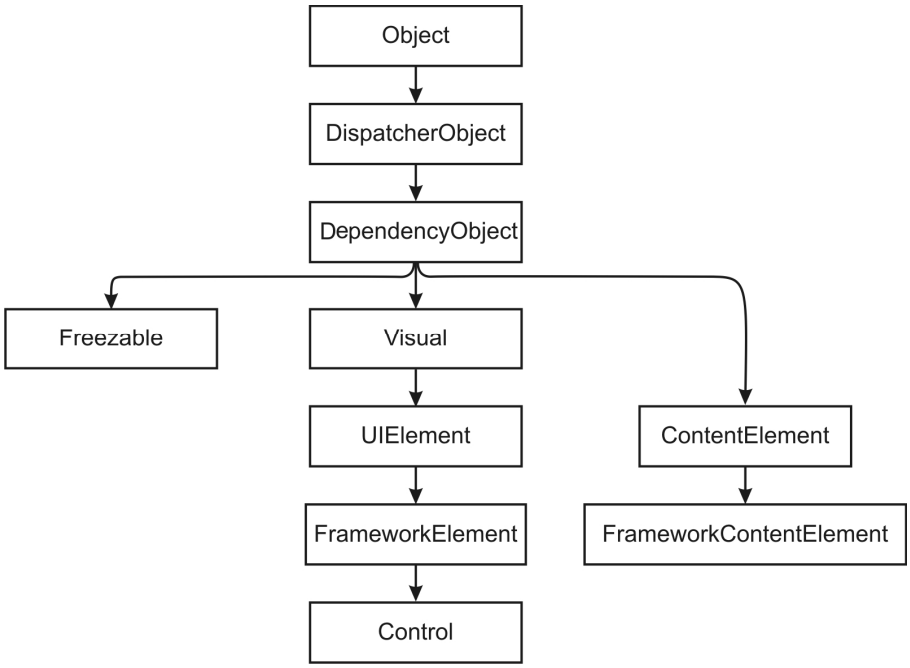

Figure C.2: Showing the Hierarchy of WPF

As seen in Figure C.2, the System.Object,System.Threading.DispatcherObject, System.Windows.DependencyObject, System.Windows.Media.Visual, System.Windows.UIElement, System.Windows.FrameworkElement, and System.Windows.Controls.Control classes form the hierarchy of the WPF Presentation Framework. These classes are the managed components of WPF.

The `milcore` component in WPF is developed by using unmanaged code. It is capable of communicating with `DirectX` easily and acts as a medium or interface between DirectX and CLR (and managed WPF components). As `DirectX` is based on the unmanaged code, the question that arises is how the unmanaged and managed code communicates with each other. WPF uses a number of classes to create communication between managed and unmanaged code. The class hierarchy of WPF is shown in Figure C.3:

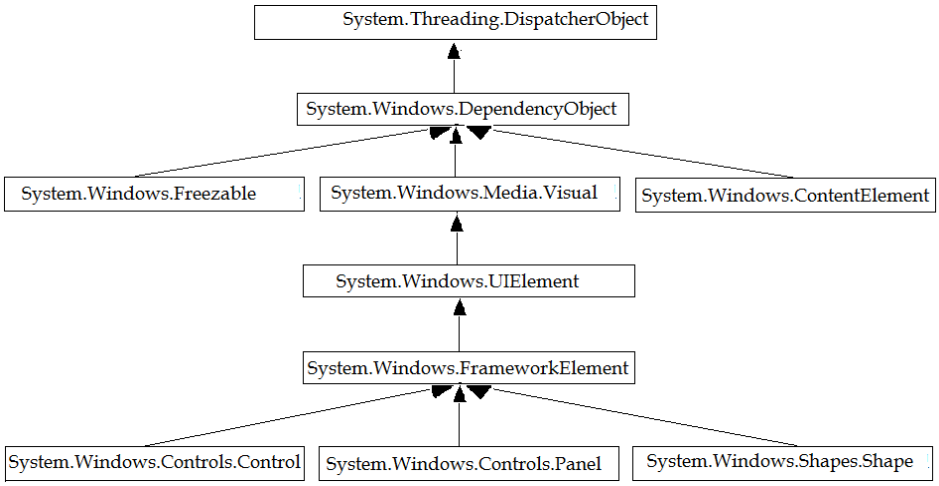

Figure C.3: Showing the Class Hierarchy of WPF

Now, let's discuss the following major classes of WPF:

- `System.Threading.DispatcherObject`—Manages the threads and concurrency of tasks. The inheritance hierarchy of the `DispatcherObject` class is as follows:

```
System.Object
  System.Windows.Threading.DispatcherObject
```

- `System.Windows.DependencyObject`—Provides properties to a user object, such as buttons. The inheritance hierarchy of the `DependencyObject` class is as follows:

```
System.Object
  System.Windows.Threading.DispatcherObject
        System.Windows.DependencyObject
```

- `System.Windows.Media.Visual`—Provides rendering support in WPF, which includes coordinate transformation. This class is the main entry point for communication between managed application programming interface (API) and unmanaged code, `milcore`. The Visual class in WPF is provided as a public abstract class from which further classes are derived. Figure C.4 shows the hierarchy of class of the `Visual` class:

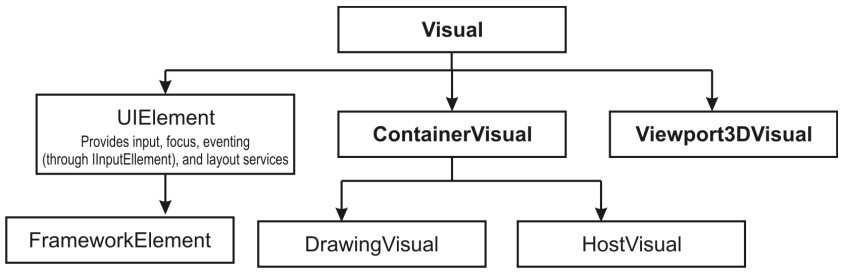

Figure C.4: Showing the Visual Class Hierarchy

The inheritance hierarchy of the `Visual` class is as follows:

```
System.Object
  System.Windows.Threading.DispatcherObject
        System.Windows.DependencyObject
              System.Windows.Media.Visual
```

- `System.Windows.UIElement`—Defines the layouts and events for UI. The inheritance hierarchy of the `UIElement` class is as follows:

```
System.Object
  System.Windows.Threading.DispatcherObject
```

```
            System.Windows.DependencyObject
                    System.Windows.Media.Visual
                            System.Windows.UIElement
```

- `System.Windows.FrameworkElement`—Provides WPF framework-level element classes and the WPF core-level set of the `UIElement` class presentation services. It consists of high-level managed classes that line-of-business application developers often use to create a WPF application. The inheritance hierarchy of the `FrameworkElement` class is as follows:

```
System.Object
  System.Windows.Threading.DispatcherObject
        System.Windows.DependencyObject
                System.Windows.Media.Visual
                        System.Windows.UIElement
                                System.Windows.FrameworkElement
```

- `System.Windows.Controls.Control`—Manages the controls used on UI. The inheritance hierarchy of the `Control` class is as follows:

```
System.Object
  System.Windows.Threading.DispatcherObject
        System.Windows.DependencyObject
                System.Windows.Media.Visual
                        System.Windows.UIElement
                                System.Windows.FrameworkElement
                                        System.Windows.Controls.Control
```

Now, let's discuss the features of WPF.

Features of WPF

The goal of introducing WPF is to provide a UI that integrates capabilities of previous frameworks including Graphical Device Interface (GDI), GDI+, and HyperText Markup Language (HTML). WPF uses the principle of managed programming model as the programmable platform. It supports XAML language, which is a rich, consistent, and complete markup- based language. WPF provides the following advanced UI functionalities other than the basic functions of designing and programming:

- **Documents**—Provides support for XML Paper Specification (XPS) and flow documents. XPS documents are supported by the XAML `FixedDocument` tag. You can also use the `FlowDocument` tag for displaying the flow documents. XPS documents are the documents that are based on XML specifications used by Windows Vista. Flow documents are those documents that are used for optimizing the content according to the runtime variables, such as windows size and device resolution.
- **Graphics**—Provides support for 2-D and 3-D graphics. It provides brushes, shapes, and pens for drawing 2-D graphics. It also provides XAML-based elements for defining 3-D graphics, which require lighting and camera position. WPF does not rely on `GDI+` for `3-D` graphics.
- **Images**—Uses the image tag of XAML to support different type of image formats, such as jpeg, gif, and png. It uses `Windows Imaging Component` to provide software that displays and stores images.
- **Media**—Uses the `MediaElement` tag of XAML for displaying video and audio formats. These formats include WMV, AVI, and MPEG file types. You can use this tag with other XAML tags for displaying 3-D cubes.
- **Animation**—Provides support for animation. You can use this support for creating storyboards, including timeline animations.
- **Data Binding**—Provides support for data binding which is automatically performed in any WPF application. It also provides support for sorting and filtering of data.

  WPF forms an essential component of .NET Framework that enables .NET application developers to create WPF applications with a visually appealing UI. Using WPF 4.0, .NET developers can integrate 3D graphics, multimedia, animation, document support, and speech recognition into their applications. The following are the list of improvements that are introduced in WPF 4.0:
- **Visual State Manager**—Refers to the class introduced with WPF 4.0 that enables the .NET developers to define the appearance of a control as a result of some user interaction. In other words, the

VisualStateManager class can be used to apply different visual states to a control. For instance, if there is a button in your application, then using the Visual State Manager class, you can define how the button would appear if a user hovers the mouse over the button or presses the button. The Visual State Manager class is fully supported in Microsoft Expression Blend.

- **Addition of new controls**—Refers to the introduction of new controls that include DataGrid, Calendar, and DatePicker. These new WPF controls enable application developers to create enhanced enterprise-level applications. All the new WPF controls are almost 100% compatible with their Silverlight versions; thereby, allowing the reuse of code between Silverlight and WPF implementations.
- **Touch input and their manipulation**—Refers to the support for touch input that WPF 4.0 elements provide. UIElement, UIElement3D, and ContentElement are classes, which consist of events that are raised when you touch an element on a touch-screen. You can manipulate the UIElement class by rotating, scaling, or translating it. For instance, when you view some photographs in an application, you can zoom, move, resize, or rotate the photograph on a computer screen.
- **New text rendering stack**—Refers to the improvements made in rendering text, which include text clarity, configurability, and support for international languages. Moreover, users can now select from auto, aliased, grayscale, or ClearType text rendering modes. In addition, the text stack includes support for the display-optimized character layout so that text with sharpness comparable to Win32/GDI text can be generated.
- **Improved support for deployment**—Refers to the improvements that have been made to the deployment size, time, and the overall deployment experience. A WPF application can either target the standard installation of full .NET Framework 4.0 or .NET Framework 4 Client Profile, which is a subset of full .NET Framework 4.0. The overall application deployment experience is improved when WPF applications target .NET Framework 4 Client Profile, as it contains only those functionalities that are required by desktop applications, such as Windows Forms and WPF applications. Therefore, instead of targeting full .Net Framework 4.0, desktop applications generally target .NET Framework 4 Client Profile by default.
- **Graphics and animation**—Includes new features such as layout rendering, cached composition, pixel shader 3 support, Easing functions. The previously mentioned features provide enhanced effects in WPF.
- **XAML Browser applications**—Includes two new features such as HTML-XBAP Script Interop and Full-Trust XBAP Deployment. The HTML-XBAP Script Interop allows you to communicate with the Web page that contains the XBAP during the hosting of application in a frame.
- **WPF and Windows**—Communicates with each other more efficiently as Windows 7 provides functionality that allows the user to use the taskbar button to communicate status to a user and expose common tasks.
- **WPF and Silverlight Designer**—Includes new features such as support for silverlight, support for multiple platform versions, visual databinding, auto layout, and improved property editing.

Next, let's discuss how to create XAML browser applications in WPF.

XAML Browser Applications

XAML is a declarative markup language that is used to create UIs for WPF and Silverlight applications. An XAML file is an XML-based file that has the `.xaml` extension. Using XAML files, you can create UI elements in the XAML markup, separate the UI from the application logic and join markup with the UI elements through partial classes. In addition, XAML can instantiate the managed objects that help to simplify the code and enables debugging access for objects, which are created in XAML. An XAML browser application (XBAP, pronounced as *x-bap*) is a Web server hosted WPF application that runs in a Web browser.

If you create an XBAP application in WPF 4.0, then you can communicate with the Web page in which your XBAP application is hosted. However, by default, an XBAP application cannot access the system resources. Therefore, to overcome this, if the user allows then, an XBAP application can be installed as a full trust application, which means, the application will have access to the system's resources. Prior to an XBAP application being installed as a full trust application, you receive a **ClickOnce** prompt.

NOTE

An updated version of XAML, called XAML 2009, has been introduced with WPF 4.0. XAML 2009 can be used with loose XAML files only. A loose XAML file is a markup file that defines the UI of an application and can be run on a Web browser without compiling. As a loose XAML file is not compiled with any application; therefore, you cannot use any external assemblies, classes or code-behind files in the loose XAML file. In addition, you cannot perform two-way data binding in XAML files.

Visual Studio 2010 provides a project template to create XBAP applications. Perform the following steps to create a XBAP application:

1. Select File→New→Project from the menu bar. The New Project dialog box appears.
2. Select either `Visual C#` or `Visual Basic` from the `Installed Templates` pane in the `New Project` dialog box.
3. Select WPF Browser Application template from the middle pane on the New Project dialog box (Figure C.5).
4. Enter an appropriate name and location for the application in the Name text box and Location combo box, respectively (Figure C.5). In this case, we have provided the name as MyWPFBrowserApplication.
5. Click the OK button, as shown in Figure C.5:

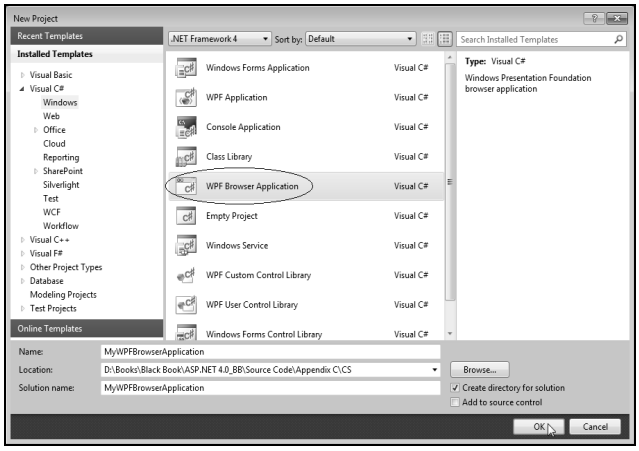

Figure C.5: Selecting WPF Browser Application

The XBAP application, MyWPFBrowserApplication, is created, as shown in Figure C.6:

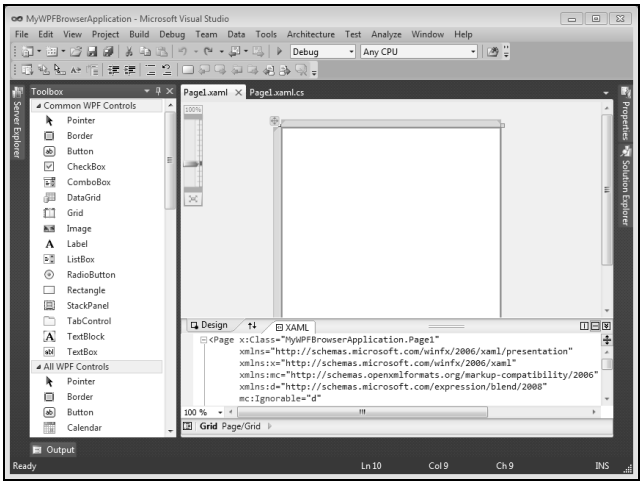

Figure C.6: Displaying a WPF Browser Application

6. Drag a button control to the design surface or add the following code snippet in the XAML code:

```
<Grid>
        <Button Height="34" Margin="10,10,0,0" Name="button1" VerticalAlignment="Top"
  Click="button1_Click" Cursor="Pen" HorizontalAlignment="Left" Width="133">Click
  Me</Button>
    </Grid>
```

The designer surface application appears, as shown in Figure C.7:

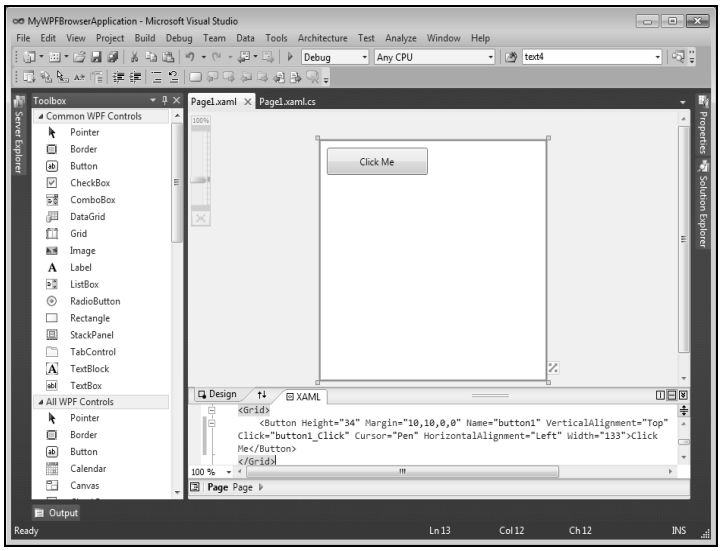

Figure C.7: Adding a Button Control

You can find the code as MyWPFBrowserApplication in C# and MyWPFBrowserApplicationVB in VB on the CD.

7. Add the following code snippet to the Page1.xaml.cs file in the click event of the button, `button1`:

In C#

```
button1.Content = " Clicked";
```

In VB

```
button1.Content = " Clicked"
```

In the preceding code snippet, you can note the difference of UI properties between WPF applications and XBAP applications, such as in the preceding code snippet, the `Content` property of button, button1, is used instead of `Text` property to display the text.

8. Run the `MyWPFBrowserApplication` application by pressing the F5 key. The output appears, as shown in Figure C.8:

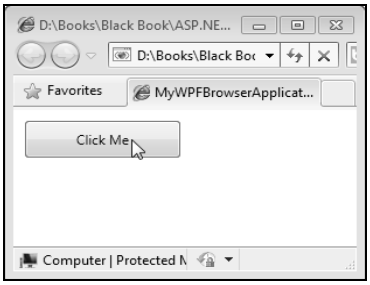

Figure C.8: Displaying the WPF Application in Web Browser

9. Click the `Click Me` button. The output appears, as shown in Figure C.9:

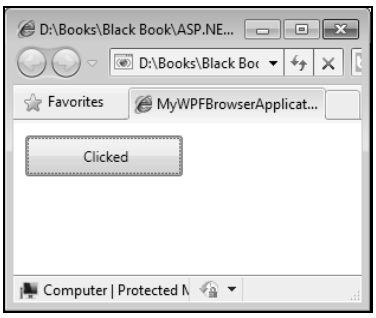

Figure C.9: Displaying the Changed Text of the Button

To understand other functionalities of WPF, such as typography, create a XBAP application and save it as `WPFXbapApp`. You can find the code of `WPFXbapApp` application as `WPFXbapApp` and `WPFXbapAppVB` in C# and VB, respectively, on the CD. Replace the code of the Page1.xaml file with the code, shown in Listing C.1, to add the required controls to the application and set their properties:

Listing C.1: Showing the Code for the Page1.xaml File

```
<Page x:Class="WPFXbapApp.Page1"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      mc:Ignorable="d"
      d:DesignHeight="300" d:DesignWidth="300"
      Title="Page1">
    <Grid>
        <Rectangle Canvas.Left="-526.614" Canvas.Top="-159.984" Height="92.5"
           VerticalAlignment="Top">
            <Rectangle.Fill>
                <LinearGradientBrush EndPoint='0,1'>
                    <LinearGradientBrush.GradientStops>
                        <GradientStop Offset='0' Color='#FFFF0000' />
                        <GradientStop Offset='.39' Color='#9900FF00' />
                        <GradientStop Offset='.16' Color='#FF0000FF' />
                        <GradientStop Offset='.45' Color='#00FFFFFF' />
                    </LinearGradientBrush.GradientStops>
                </LinearGradientBrush>
            </Rectangle.Fill>
        </Rectangle>
        <Button Margin="97.5,143.75,57.5,88.75" Name="button1" Click="button1_Click">
            <Button.LayoutTransform>
                <ScaleTransform ScaleX="5" ScaleY="5" />
            </Button.LayoutTransform>
            <StackPanel Orientation="Horizontal">
                <Canvas Width="20" Height="18" VerticalAlignment="Center">
                    <Ellipse Canvas.Left="1" Canvas.Top="1" Width="18" Height="18"
                         Fill="CadetBlue" Stroke="Black" />
                    <Ellipse Canvas.Left="5.5" Canvas.Top="5" Width="2.5" Height="3"
                         Fill="Black" />
                    <Ellipse Canvas.Left="11" Canvas.Top="5" Width="2.5" Height="3"
                         Fill="Black" />
                    <Path Data="M 5,10 A 3,3 0 0 0 13,10" Stroke="Black" />
                </Canvas>
                <TextBlock VerticalAlignment="Center">Click Me!</TextBlock>
            </StackPanel>
        </Button>
    </Grid>
</Page>
```

NOTE

The code for the Page1.xaml file is same for both C# and VB. However, the value of the x:Class attribute in the Page tag of WPF browser application changes according to the name that you give to your application. For instance, if the name of your application is MyWPFApplication, then the value of x:Class attribute will be MyWPFApplication.Page1.

After adding the code given in Listing C.1, add the code, shown in Listing C.2, in the code-behind file of the `Page1.xaml` file:

Listing C.2: Showing the Code for the Code-Behind File of the `Page1.xaml` File

In VB

```
Imports System.Threading
Class Page1

Private Sub button1_Click(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs)
        Dim text As New FormattedText("KOGENT SOLUTIONS!",
           Thread.CurrentThread.CurrentUICulture, FlowDirection.LeftToRight, New
Typeface("Arial Black"), 10, Brushes.Black)
        Dim textGeometry As Geometry = Text.BuildGeometry(New Point(0, 0))
        button1.Clip = textGeometry
    End Sub
End Class
```

In C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Threading;

namespace WPFXbapApp
{
    /// <summary>
    /// Interaction logic for Page1.xaml
    /// </summary>
    public partial class Page1 : Page
    {
        public Page1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, RoutedEventArgs e)
        {
            FormattedText text = new FormattedText("KOGENT SOLUTIONS!",
Thread.CurrentThread.CurrentUICulture, FlowDirection.LeftToRight, new
              Typeface("Arial Black"), 10, Brushes.Black);
            Geometry textGeometry = text.BuildGeometry(new Point(0, 0));
            button1.Clip = textGeometry;
        }
    }
}
```

Now, let's run the `WPFXbapApp` application by pressing the F5 key. The output appears, as shown in Figure C.10:

Figure C.10: Displaying the Output of the WPFXbapApp Application

In Listing C.2, you can see that there is no attached image for the button to display the smiley on the button. By using XAML, you can draw any image on the page. As shown in Figure C.10, at the top of the page, a rectangle is filled with combination of gradients. When you click the `Click Me!` button, the text changes, as shown in Figure C.11:

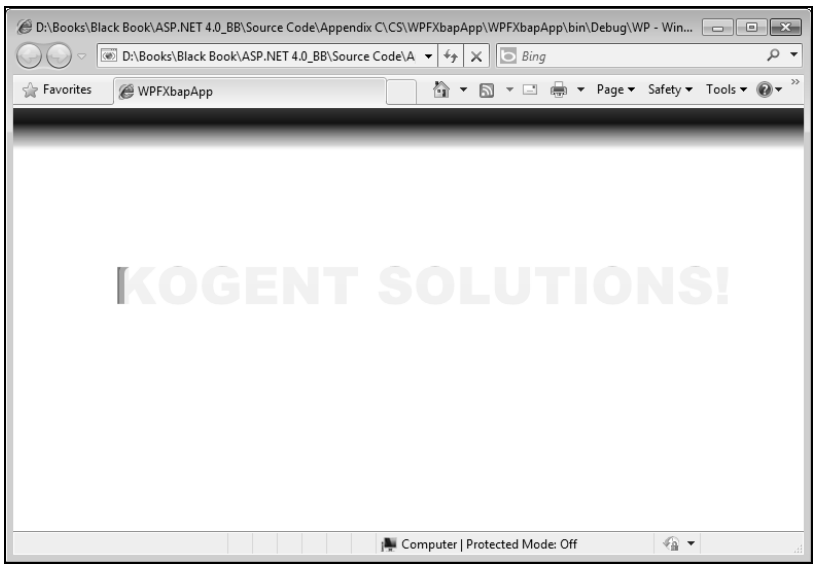

Figure C.11: Displaying the Output after the Click Me! Button is Clicked

Now, let's see how to create animated objects by using WPF and XML. To create animated objects, add a page in the `WPFXbapApp` application, WPF Browser application (In the Solution Explorer, right-click `WPFXbapApp` →Add→New Item. Select WPF→ Page (WPF)) option from the Add New Item dialog box and name it as Page2.xaml. Replace the code of the Page2.xaml file with the code, shown in Listing C.3, to add the required controls and set their properties:

Listing C.3: Showing the Code for the Page2.xaml File

```
<Page x:Class="WpfBrowserApplication1.Page2"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Page2">
  <Grid>
        <Ellipse Fill="BlueViolet" Width="25" Height="120">
        <Ellipse.Triggers>
        <EventTrigger RoutedEvent="Ellipse.MouseEnter">
        <BeginStoryboard Name="Animenter">
        <Storyboard>
        <DoubleAnimation By="400" Duration="0:0:4"
        Storyboard.TargetProperty="Width" />
        <ColorAnimation Storyboard.TargetProperty="(Ellipse.Fill).(SolidColorBrush.Color)"
          Duration="0:0:7" From="BlueViolet" To="Red" RepeatBehavior ="Forever"
          AutoReverse="True" /> </Storyboard>
        </BeginStoryboard>
        </EventTrigger>
        <EventTrigger RoutedEvent="Ellipse.MouseLeave">
        <BeginStoryboard HandoffBehavior="Compose" Name="Animleave">
        <Storyboard>
        <DoubleAnimation By="-400" Duration="0:0:4"
        Storyboard.TargetProperty="Width" />
        </Storyboard>
        </BeginStoryboard>
        </EventTrigger>
        <EventTrigger RoutedEvent="Ellipse.Unloaded">
        <RemoveStoryboard BeginStoryboardName="Animenter" />
        <RemoveStoryboard BeginStoryboardName="Animleave" />
        </EventTrigger>
        </Ellipse.Triggers>
        </Ellipse>
  </Grid>
</Page>
```

Double-click `App.xaml` file in Solution Explorer to open it and set the `StartupUri` property to `Page2.xaml`. Now, run the `WPFXbapApp` application by pressing the F5 key. The output appears, as shown in Figure C.12:

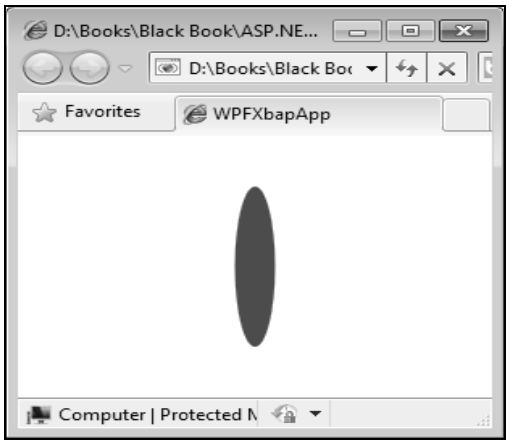

Figure C.12: Displaying the Output after Adding Page2.xaml

Figure C.12 displays an animated ellipse, which increases in width when you place the mouse-pointer over it, as shown in Figure C.13:

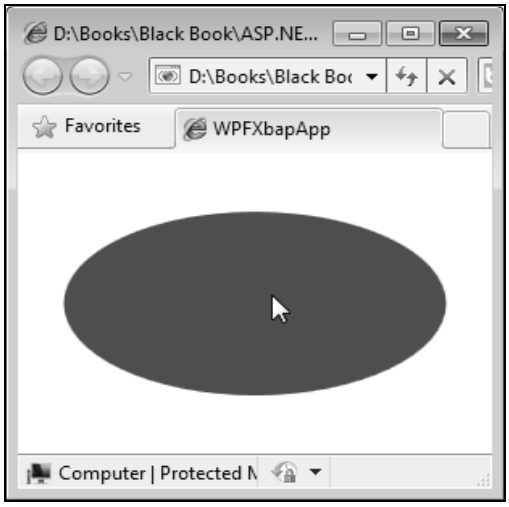

Figure C.13: Increasing Width of Ellipse

Moreover, along with the increase in width, the color of the ellipse also changes from blue to red through `color animation`. When you move the mouse-pointer out of the ellipse, its width start decreasing, as shown in Figure C.14:

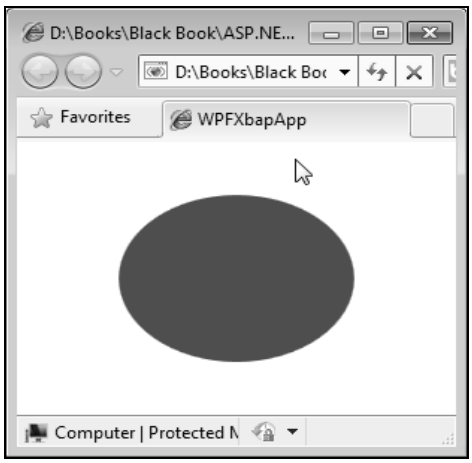

Figure C.14: Decreasing Width of Ellipse

Now, let's discuss the new controls introduced in WPF 4.0.

New Controls in WPF 4.0

As discussed, WPF offers a unified programming model to create applications that have a separate UI and business logic. In addition, WPF has a rich set of controls that enables you to develop high-end applications. Some of the WPF controls are similar to those in Windows Forms, such as the Button and TextBox controls. However, certain WPF controls, such as Grid and Canvas, are unique in terms of their appearance and behavior. Moreover, controls, such as DataGrid, Calendar, and DatePicker, are the ones that have been introduced in WPF 4.0. The DataGrid control is a data display control used to display some information from a data source, such as a SQL Server database or an XML file. The Calendar and DatePicker controls are date display and selection

controls that can be used to display some dates and select them on a calendar. Let's discuss about these three controls one by one, starting with the DataGrid control.

The DataGrid Control

DataGrid is a new control that has been introduced in WPF 4.0. Similar to the DataGrid control in Windows Forms and ASP.NET Web Forms, the DataGrid control in WPF 4.0 is used to display data in a customizable grid. The data inside a DataGrid control is displayed in the form of rows and columns. You can display the data from a database inside the DataGrid control or from any collection that implements the IEnumerable collection. The DataGrid control is found in the System.Windows.Controls namespace, which is present in the PresentationFramework.dll assembly.

The DataGrid control is an instance of the DataGrid class, which has various properties that allow you to work with the DataGrid control. Table C.1 lists the noteworthy properties of the DataGrid class:

Table C.1: Noteworthy Properties of the DataGrid Class

Property	Description
AlternatingRowBackground	Retrieves or sets the back color for alternating rows in a DataGrid control
AutoGenerateColumns	Retrieves or sets a value indicating whether or not the columns in a DataGrid control are generated automatically
CanUserAddRows	Retrieves or sets a value indicating whether or not a user can add new rows to a DataGrid control
CanUserDeleteRows	Retrieves or sets a value indicating whether or not a user can delete rows from a DataGrid control
CellStyle	Retrieves or sets the style applied to all the cells in a DataGrid control
Columns	Retrieves a collection that contains all the columns in a DataGrid control
HasItems	Retrieves a value that indicates whether the ItemsControl control contains items. The ItemsControl control represents a control that can be used to present a collection of items.
Items	Retrieves a collection used to generate the content of the ItemsControl control
ItemTemplate	Retrieves or sets the DataTemplate objects used to display item
SelectedCells	Retrieves a list of cells that are currently selected
SelectedIndex	Retrieves or assigns the index of the first item in the current selection or returns negative one (-1) if no item is selected
SelectedItem	Retrieves or assigns the first item in the current selection or returns null if no item is selected
SelectedValue	Retrieves or sets the value of the SelectedItem property
Template	Retrieves or sets a control template

Now, let's perform the following steps to create a XBAP application to learn how to use a DataGrid control.

1. Repeat steps 1 to 2 as followed earlier while creating the MyWPFBrowserApplication in the XAML Browser Applications section of this chapter.
2. Enter the name DataGridDemo in the Name text box and specify an appropriate location for the application in the Location combo box.
3. Click the OK button. The DataGridDemo application has been created. You can find the code for this application as DataGridDemo and DataGridDemoVB in C# and VB, respectively, on the CD.
4. Add the code, given in Listing C.4, in the Page1.xaml file:

Listing C.4: Using a DataGrid Control

```
<Page x:Class="DataGridDemo.Page1"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        mc:Ignorable="d"
        d:DesignHeight="300" d:DesignWidth="300"
        Title="Page1">
    <Grid Height="300" Width="300">
        <DataGrid AutoGenerateColumns="False" Height="200" HorizontalAlignment="Left"
         Margin="10,10,0,0" Name="dataGrid1" VerticalAlignment="Top"
         Width="200" AlternatingRowBackground="Beige"/>
    </Grid>
</Page>
```

In Listing C.4, a DataGrid control is added to the Grid control. The AutoGenerateColumns property of the DataGrid control is set to False and the AlternatingRowBackground property of is assigned the value, Beige. However, the DataGrid control we have added in this application does not display any data.

5. Add the code, shown in Listing C.5, to the code-behind file of the DataGridDemo application to populate the DataGrid control with data programmatically:

Listing C.5: Showing the Code to Populate a DataGrid control

In VB

```
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Input
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Navigation
Imports System.Windows.Shapes

Namespace DataGridDemo
    ''' <summary>
    ''' Interaction logic for Page1.xaml
    ''' </summary>
    Partial Public Class Page1
        Inherits Page
        Public Sub New()
            InitializeComponent()
            dataGrid1.Items.Add("Tim")
            dataGrid1.Items.Add("Jack")
End Sub
    End Class
End Namespace
```

In C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace DataGridDemo
```

```
    {
        /// <summary>
        /// Interaction logic for Page1.xaml
        /// </summary>
        public partial class Page1 : Page
        {
            public Page1()
            {
                InitializeComponent();
                dataGrid1.Items.Add("Tim");
                dataGrid1.Items.Add("Jack");
            }
        }
    }
```

6. Press the F5 key to run the DataGridDemo application. The output appears, as shown in Figure C.15:

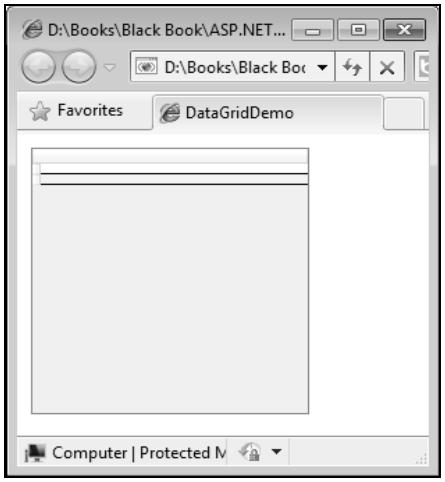

Figure C.15: Displaying a DataGrid Control

In Figure C.15, you can see a grid with two horizontal lines drawn on it. These lines correspond to two rows we have added in Listing C.5. However, no data is displayed in the DataGrid control. This is because we have not defined any columns for the control. To display data in the DataGrid control, we have to define the columns.

Table C.2 lists the four types of columns available for a DataGrid:

Table C.2: Columns Types of the DataGrid Control	
Property	**Description**
DataGridTextColumn	Displays textual data
DataGridCheckBoxColumn	Displays Boolean data
DataGridComboBoxColumn	Displays enumerations
DataGridHyperlinkColumn	Displays URIs

7. Add the following code snippet to the code-behind file of the DataGridDemo application to add a column to a DataGrid control:

In *VB*

```
    dataGrid1.Columns.Add(New DataGridTextColumn())
```

In C#

```
    dataGrid1.Columns.Add(new DataGridTextColumn());
```

8. Press the F5 key to run the DataGridDemo application. The output appears, as shown in Figure C.16:

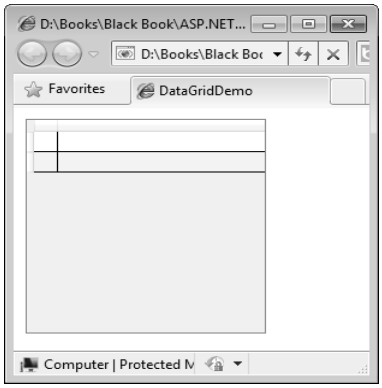

Figure C.16: Displaying a DataGrid Control after Adding Columns

In Figure C.16, a column is added to the DataGrid control; however, the data is still not displayed inside it. This is because you have to setup a binding between the objects of the collection whose data you want to display inside the DataGrid Control.

9. Add the code, given in Listing C.6, to the code-behind file of the DataGridDemo application to setup the binding:

Listing C.6: Showing the Code to Perform Binding

In VB

```
Dim col1 As New DataGridTextColumn()
            dataGrid1.Columns.Add(col1)
            col1.Binding = New Binding(".")
```

In C#

```
DataGridTextColumn col1 = new DataGridTextColumn();
dataGrid1.Columns.Add(col1);
col1.Binding = new Binding(".");
```

In Listing C.6, the Binding (".") method implies to bind the entire object col1 with the DataGrid control, and that the binding engine automatically invokes the object's ToString() method. As the objects in the collection are of String type, their values are displayed as required and finally the DataGrid control displays the data.

10. Press the F5 key to run the DataGridDemo application. The output appears, as shown in Figure C.17:

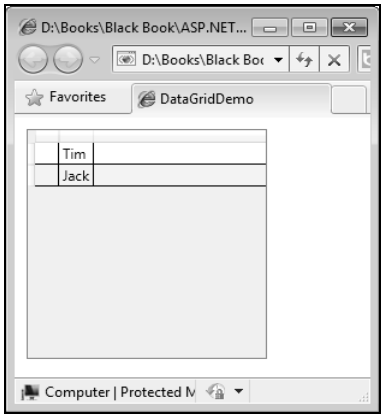

Figure C.17: Showing the Output after Setting the Binding

You can enhance the look and appearance of a DataGrid control by setting its properties given in Table C.2. You can also set the headers for the column of a DataGrid control, as shown in the following code snippet:

In VB

```
    col1.Header = "Name"
```

In C#

```
    col1.Header = "Name";
```

11. Press the F5 key to run the DataGridDemo application again. The final output appears, as shown in Figure C.18:

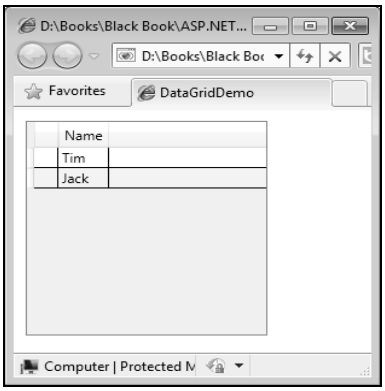

Figure C.18: Displaying the Output after Setting the Column Header

In Figure C.18, data is displayed inside the DataGrid control under the Name column. Next, let's learn how to use the Calendar control in WPF browser applications.

The Calendar Control

The Calendar control is a new control that has been introduced with the latest version of WPF. It is a visual calendar, in which you can select a particular date. Whenever a date is selected on a Calendar control, an event is fired on the selected date. The Calendar control is found in the System.Windows.Controls namespace, which is present in the PresentationFramework.dll assembly. The Calendar control inherits the Control class. Table C.3 lists some of noteworthy properties of the Control class:

Table C.3: Noteworthy Properties of Calendar Control

Property	Description
BlackOutDates	Retrieves a collection of dates that cannot be selected in a Calendar control
DisplayDate	Retrieves or sets the date to display in a Calendar control
DisplayDateEnd	Retrieves or sets the last date in the date range that is available in a Calendar control
DisplayDateStart	Retrieves or sets the first date that is available in a Calendar control
DisplayMode	Retrieves or sets a value that indicates whether a Calendar control displays a month, year, or decade
FirstDayOfWeek	Retrieves or sets the day that is considered the beginning of the week in a Calendar control
SelectedDate	Retrieves or sets the currently selected date in a Calendar control
SelectedDates	Retrieves a collection of selected dates in a Calendar control
SelectionMode	Retrieves or sets a value that indicates what kind of selections are allowed in a Calendar control

Now, let's create a WPF browser application named CalendarDemo to learn how to use the Calendar control. Perform the following steps to create the CalendarDemo application:

1. Repeat steps 1 to 2 as followed earlier while creating the MyWPFBrowserApplication in the XAML Browser Applications section of this chapter.
2. Enter the name CalendarDemo in the Name text box and specify an appropriate location to save the application in the Location combo box. You can find the code for this application as CalendarDemoCS and CalendarDemoVB in C# and VB, respectively, on the CD.
3. Click the OK button. The CalendarDemo application has been created.
4. Add the code, given in Listing C.7, in the Page1.xaml file:

Listing C.7: Using the Calendar Control

```
<Page x:Class="CalendarDemoVB.Page1"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      mc:Ignorable="d"
      d:DesignHeight="300" d:DesignWidth="300"
      Title="Page1">
    <Grid>
        <Calendar Name="myCal1" HorizontalAlignment="Center" VerticalAlignment="Center"
                  DisplayMode="Month" SelectedDatesChanged="myCal1_SelectedDatesChanged">
            <Calendar.BlackoutDates>
                <CalendarDateRange Start="5/30/2010" End="6/10/2010"></CalendarDateRange>
            </Calendar.BlackoutDates>
            <Calendar.BorderBrush>
                <LinearGradientBrush StartPoint="0,0" EndPoint="1,1" >
                    <GradientStop Color="Blue" Offset="0" />
                    <GradientStop Color="Red" Offset="1.0" />
                </LinearGradientBrush>
            </Calendar.BorderBrush>
            <Calendar.Background>
                <LinearGradientBrush StartPoint="0,0" EndPoint="1,1" >
                    <GradientStop Color="Pink" Offset="0.1" />
                    <GradientStop Color="LightGreen" Offset="0.25" />
                    <GradientStop Color="Yellow" Offset="0.75" />
                    <GradientStop Color="Pink" Offset="1.0" />
                </LinearGradientBrush>
            </Calendar.Background>
            <Calendar.Foreground>
                <LinearGradientBrush StartPoint="0,0" EndPoint="1,1" >
                    <GradientStop Color="Black" Offset="0.25" />
                    <GradientStop Color="Green" Offset="1.0" />
                </LinearGradientBrush>
            </Calendar.Foreground>
        </Calendar>
    </Grid>
</Page>
```

In Listing C.7, a Calendar control is added to a Grid control. The Name property is used to uniquely identify the Calendar control while the DisplayMode property is used to specify the format in which a calendar is to be displayed. In our case, we have set the DisplayMode property to Month. The BlackoutDates property of the Calendar control is also set to specify the range of dates that a user cannot select.

5. Add the code, given in Listing C.8, to the code-behind file of the CalendarDemo application:

Listing C.8: Adding the Code to the Code-Behind File

In VB

```
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
```

```
Imports System.Windows.Input
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Navigation
Imports System.Windows.Shapes
Namespace CalendarDemoVB
    ''' <summary>
    ''' Interaction logic for Page1.xaml
    ''' </summary>
    Partial Public Class Page1
        Inherits Page
        Public Sub New()
            InitializeComponent()
            SetDisplayDates()
        End Sub
        Private Sub SetDisplayDates()
            myCal1.DisplayDate = New DateTime(2010, 5, 1)
            myCal1.DisplayDateStart = New DateTime(2010, 5, 1)
            myCal1.DisplayDateEnd = New DateTime(2010, 8, 31)
        End Sub
        Private Sub myCal1_SelectedDatesChanged(ByVal sender As Object, ByVal e As
         SelectionChangedEventArgs)
            MessageBox.Show("You selected: " + myCal1.SelectedDate.ToString())
        End Sub
    End Class
End Namespace
```

In C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace CalendarDemoCS {
    /// <summary>
    /// Interaction logic for Page1.xaml
    /// </summary>
    public partial class Page1 : Page
    {
        public Page1()
        {
            InitializeComponent();
            SetDisplayDates();
        }
        private void SetDisplayDates()
        {
            myCal1.DisplayDate = new DateTime(2010, 5, 1);
            myCal1.DisplayDateStart = new DateTime(2010, 5, 1);
            myCal1.DisplayDateEnd = new DateTime(2010, 8, 31);
        }
        private void myCal1_SelectedDatesChanged(object sender, SelectionChangedEventArgs
         e)
        {
                MessageBox.Show("You selected: " + myCal1.SelectedDate.ToString());
            }
        }
    }
```

6. Press the F5 key to run the CalendarDemo application. The output appears, as shown in Figure C.19:

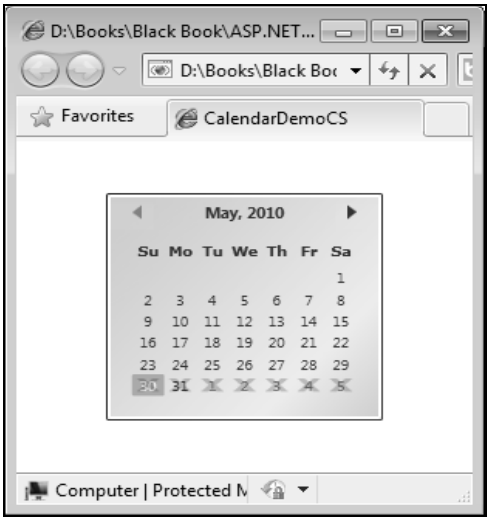

Figure C.19: Displaying a Calendar Control

In the Figure C.19, you can see that the Calendar control displays dates starting from May 1, 2010. Note that the left arrow on top of the Calendar control is disabled. In addition to this, note the Calendar control displays dates till the month of August 2010, which is the date specified in the DispalyDateEnd property.

7. Select any date from the calendar in the CalendarDemo application. A message box appears displaying the selected date (Figure C.20).
8. Click the OK button to close the message box, as shown in Figure C.20:

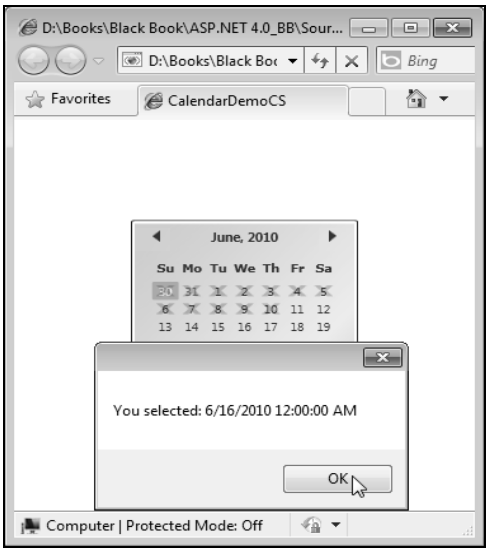

Figure C.20: Displaying the Output after Selecting a Date

Next, let's discuss about the DatePicker control in WPF 4.0.

The DatePicker Control

Similar to the Calendar control, a WPF DatePicker control also enables you to select a date. This control is located in the System.Windows.Controls namespace and inherits the DatePicker class.

Table C.4 lists the noteworthy properties of the DatePicker class:

Table C.4: Noteworthy Properties of DatePicker Class

Property	Description
BlackOutDates	Retrieves or assigns a collection of dates that are marked as not selectable in a DatePicker control
DisplayDate	Retrieves or sets the date to display in a DatePicker control
DisplayDateEnd	Retrieves or sets the last date to be displayed in a DatePicker control
DisplayDateStart	Retrieves or sets the first date to be displayed in a DatePicker control
FirstDayOfWeek	Retrieves or sets the day that is considered the beginning of the week in a DatePicker control
SelectedDate	Retrieves or sets the currently selected date in a DatePicker control
SelectedDateFormat	Retrieves or sets the format used to display the selected date in a DatePicker control

Now, let's create a WPF browser application named DatePickerDemo to learn how to use the DatePicker control. Perform the following steps to create the DatePickerDemo application:

1. Repeat steps 1 to 2 as followed earlier while creating the MyWPFBrowserApplication in the XAML Browser Applications section of this chapter.
2. Enter the name DatePickerDemo in the Name text box and specify an appropriate location to save the application in the Location combo box. You can find the code for this application as DatePickerDemoCS and DatePickerDemoVB in C# and VB, respectively, on the CD.
3. Click the OK button. The DatePickerDemo application has been created.
4. Add the code, given in Listing C.9, in the Page1.xaml file:

Listing C.9: Using the DatePicker Control

```
<Page x:Class="DatePickerDemoCS.Page1"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      mc:Ignorable="d"
      d:DesignHeight="300" d:DesignWidth="300"
      Title="Page1">
    <Grid>
        <DatePicker Name="myDP" HorizontalAlignment="Center" VerticalAlignment="Top">
            <DatePicker.BlackoutDates>
                <CalendarDateRange Start="6/1/2010" End="6/10/2010"/>
            </DatePicker.BlackoutDates>
        </DatePicker>
    </Grid>
</Page>
```

In Listing C.9, a DatePicker control is added to a Grid control. The Name property is used to uniquely identify the DatePicker control and the BlackoutDates property is used to specify the range of dates that cannot be selected by a user.

5. Add the code, given in Listing C.10, to the code-behind file of the DatePickerDemo application:

Listing C.10: Showing the Code of the Code-Behind File

In VB

```
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text
```

```
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Input
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Navigation
Imports System.Windows.Shapes

Namespace DatePickerDemoVB
    ''' <summary>
    ''' Interaction logic for Page1.xaml
    ''' </summary>
    Partial Public Class Page1
        Inherits Page
        Public Sub New()
            InitializeComponent()
            SetDisplayDates()
        End Sub
        Private Sub SetDisplayDates()
            myDP.DisplayDate = New DateTime(2010, 6, 1)
            myDP.DisplayDateStart = New DateTime(2010, 6, 1)
            myDP.DisplayDateEnd = New DateTime(2010, 6, 30)
            myDP.FirstDayOfWeek = DayOfWeek.Tuesday
        End Sub
    End Class
End Namespace
```

In C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace DatePickerDemoCS
{
    /// <summary>
    /// Interaction logic for Page1.xaml
    /// </summary>
    public partial class Page1 : Page
    {
        public Page1()
        {
            InitializeComponent();
            SetDisplayDates();
        }
        private void SetDisplayDates()
        {
            myDP.DisplayDate = new DateTime(2010, 6, 1);
            myDP.DisplayDateStart = new DateTime(2010, 6, 1);
            myDP.DisplayDateEnd = new DateTime(2010, 6, 30);
            myDP.FirstDayOfWeek = DayOfWeek.Tuesday;
        }
    }
}
```

As you can see in Listing C.10, the DisplayDate property of the DatePicker control is set to June 1, 2010. Using the DisplayDateEnd property, the last date to be displayed on the DatePicker control is set to June 30, 2010. In addition, Tuesday is assigned as the first day of the week, using the FirstDayOfWeek property.

6. Press the F5 key to run the DatePickerDemo application. The output appears (Figure C.21).
7. Click the drop-down arrow on the right-hand side of the DatePicker control, as shown in Figure C.21:

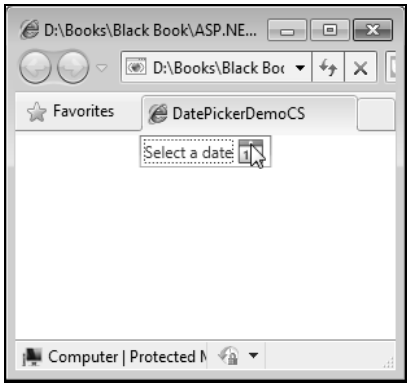

Figure C.21: Displaying a DatePicker Control

A calendar appears, as shown in Figure C.22:

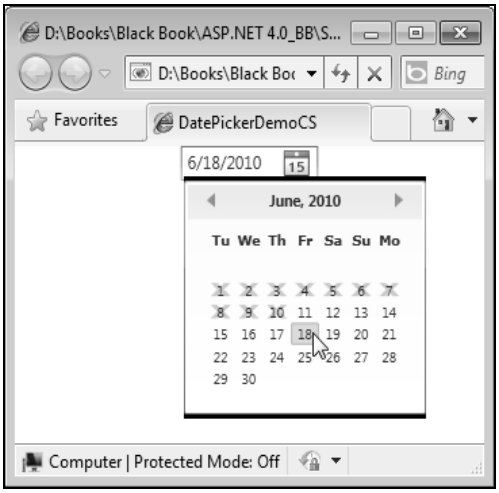

Figure C.22: Selecting a Date in a DatePicker Control

As you can see in Figure C.22, the text box above the calendar in the DatePicker control displays the date that you select in a DatePicker control.

Next, let's discuss how to perform data binding in WPF applications.

Data Binding in Windows Presentation Foundation

As discussed, WPF makes it easy to design robust and visually appealing UIs. The added capability that WPF provides is data binding. In WPF, you can perform data binding using the Framework Code technique (the code-behind file in case of WPF applications), XAML, or a combination of both.

As in other applications, such as Windows Forms and the ASP.NET Web applications, WPF also needs to have a target and a source for data binding. You can select the public property of a control in WPF to bind a data, which

includes properties of other controls, or any data source, such as the `Northwind` database. Therefore, you can say that data binding is one of the most powerful features included in WPF.

WPF data binding supports different types of binding modes between the target and the source. The data flow in a binding can be from the target to the source or vice-versa. You should specify the mode of data binding while binding the data in a WPF application. The `Mode` property defines the binding mode that determines how the data flows between the source and the target. There are the following four types of binding modes available in WPF:

- `OneTime` data binding
- `OneWay` data binding
- `TwoWay` data binding
- `OneWayToSource` data binding

Let's discuss each binding mode in detail.

OneTime Data Binding

In `OneTime` binding, the data flows from a source to a target. This binding occurs only once when an application starts or the data context changes. You should use `OneTime` binding when your data source does not implement the `INotifyPropertyChanged` interface. For example, when you do not want to change the data or add any other data to your database, you can use `OneTime` binding in your application. The `OneTime` binding can only retrieve the data, but cannot update data in your database.

OneWay Data Binding

In `OneWay` data binding also, the data flows from a source to a target. This type of binding is useful for read-only data as it does not allow changing the data from the user interface. The `OneWay` binding mode in WPF is the default binding mode.

TwoWay Data Binding

The data in `TwoWay` data binding moves in both directions, that is, from a source to a target and vice-versa. In `TwoWay` data binding, you can make changes to the data in UI. In this binding, the data is sent to the target, and if there is any change in the target property value, it will be sent back to the source. You can use the `TwoWay` binding when you want to change the data in UI which is reflected in the data source.

OneWayToSource Data Binding

In `OneWayToSource` data binding, if the target property changes, the source property also changes automatically. You can use the `OneWayToSource` binding when you want to change the data and get it updated in the source.

Next, let's learn about the sources of data binding in WPF.

Sources of Data Binding

In data binding, the sources refer to the objects from where you obtain data. WPF supports four types of binding sources: CLR object, ADO.NET data, XML data, and `DependencyObject`. In CLR object, the data binding works as long as the binding engine is able to access the source property using reflection. The XML data fails to bind when it does not have permission to access the given data. You can always bind data to an ADO.NET object and `DependencyObject`.

Now, let's learn about binding to various sources one by one.

CLR Objects

In WPF, you can bind data to public properties or the sub-properties of any CLR object. The binding object in WPF uses CLR reflection to retrieve the values of the properties. When you use the CLR object for data binding in WPF, you should implement the `INotifyPropertyChanged` interface. This interface helps you to update a target when a source property changes. This helps in ensuring that the data used in binding stays updated.

If the source object implements a proper notification mechanism, the target updates automatically. You can also use the `UpdateTarget` method to update the target property to provide property change notification.

When you bind a WPF application to a CLR object, you need to define the properties of the fields that you use in your application. You need to add a class to your WPF application to bind your data to the application. A CLR object needs to implement the `INotifypropertyChanged` interface to implement binding in WPF.

Now, let's create a WPF browser application to implement CLR object data binding. Perform the following steps to create the WPF browser application:

1. Repeat steps 1 to 2 as followed earlier while creating the MyWPFBrowserApplication application in the XAML Browser Applications section of this chapter.
2. Enter the name BindingtoCLRObject in the Name text box and specify an appropriate location to save the application in the Location combo box. You can find the code for this application as BindingtoCLRObjectCS and BindingtoCLRObjectVB in C# and VB, respectively, on the CD.
3. Click the OK button. The BindingtoCLRObject application has been created.
4. Add the code, given in Listing C.11, in the Page1.xaml file:

Listing C.11: Preparing the Interface

```
<Page x:Class="BindingtoCLRObjectCS.Page1"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      mc:Ignorable="d"
      d:DesignHeight="300" d:DesignWidth="300"
      Title="Page1">
    <Grid Name="nameGrid" Margin="5,5,5,5" >
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="30*" />
            <ColumnDefinition Width="70*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="50" />
            <RowDefinition Height="50" />
            <RowDefinition Height="50" />
            <RowDefinition Height="50" />
            <RowDefinition Height="50" />
        </Grid.RowDefinitions>
        <Label Grid.Column="0" Grid.Row="0">Name:</Label>
        <TextBox Width="200" Height="30" Grid.Column="1" Grid.Row="0"
          Text="{Binding Name}" />
        <Label Grid.Column="0" Grid.Row="1">Age:</Label>
        <TextBox Width="200" Height="30" Grid.Column="1" Grid.Row="1"
          Text="{Binding Age}" />
        <Label Grid.Column="0" Grid.Row="2">Profile:</Label>
        <TextBox Width="200" Height="30" Grid.Column="1" Grid.Row="2"
          Text="{Binding Profile}" />
        <Label Grid.Column="0" Grid.Row="3">EmployeeID:</Label>
        <TextBox Width="200" Height="30" Grid.Column="1" Grid.Row="3"
         Text="{Binding EmployeeID}" />
        <Button Margin="5,5,5,5" Grid.Column="1" Grid.Row="4" Name="Button"
          Click="Button_Click">SHOW DATA</Button>
    </Grid>
</Page>
```

After adding the preceding code to the Page1.xaml file, add a class to your application. Perform the following steps to add a class:

5. Right-click the name of your application in Solution Explorer and select `Add→Class`, as shown in Figure C.23:

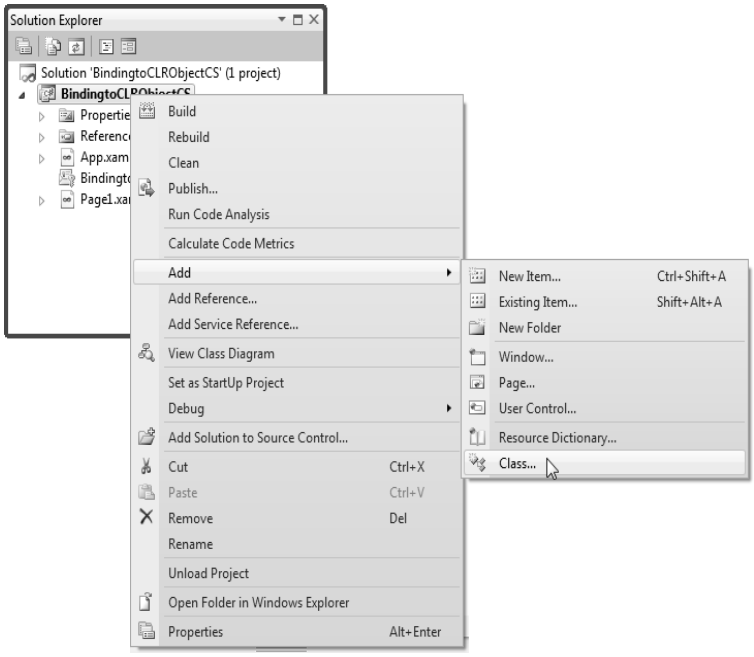

Figure C.23: Adding a Class

The Add New Item dialog box appears.

6. Enter the name `Data` in the Name text box and click the OK button.
7. Add the code, shown in Listing C.12, to the `Data` class:

Listing C.12: Adding the Code to the Data Class

In VB

```
    Imports System
    Imports System.Collections.Generic

    Namespace BindingtoCLRObject
        Friend Class Data
            Public Sub New(ByVal name As String, ByVal age As String, ByVal profile As String,
             ByVal id As String, ByVal ParamArray names As String())
                Me.name_Renamed = name
                Me.age_Renamed = age
                Me.profile_Renamed = profile
                Me.id = id
                For Each Employee As String In names
                    Me.names_Renamed.Add(name)
                Next Employee
            End Sub
            Public Sub New()
                Me.New("", "", "", "")
            End Sub
            Private name_Renamed As String
            Public Property Name() As String
                Get
                    Return name_Renamed
                End Get
                Set(ByVal value As String)
                    name_Renamed = value
                End Set
            End Property
```

```
            Private age_Renamed As String
            Public Property Age() As String
                Get
                    Return age_Renamed
                End Get
                Set(ByVal value As String)
                    age_Renamed = value
                End Set
            End Property
            Private profile_Renamed As String
            Public Property Profile() As String
                Get
                    Return profile_Renamed
                End Get
                Set(ByVal value As String)
                    profile_Renamed = value
                End Set
            End Property
            Private id As String
            Public Property EmployeeID() As String
                Get
                    Return id
                End Get
                Set(ByVal value As String)
                    id = value
                End Set
            End Property
            Public Overrides Function ToString() As String
                Return name_Renamed
            End Function
            Private ReadOnly names_Renamed As List(Of String) = New List(Of String)()
            Public ReadOnly Property Names() As String()
                Get
                    Return names_Renamed.ToArray()
                End Get
            End Property

        End Class
    End Namespace
```

In C#

```
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;

    namespace BindingtoCLRObjectCS
    {
        class Data
        {
            public Data(string name, string age, string profile, string id,
             params string[] names)
            {
                this.name = name;
                this.age = age;
                this.profile = profile;
                this.id = id;
                foreach (string Employee in names)
                {
                    this.names.Add(name);
                }
            }
            public Data()
                : this("", "", "", "")
            {
```

```
            }
            private string name;
            public string Name
            {
                get { return name; }
                set { name = value; }
            }
            private string age;
            public string Age
            {
                get { return age; }
                set { age = value; }
            }
            private string profile;
            public string Profile
            {
                get { return profile; }
                set { profile = value; }
            }
            private string id;
            public string EmployeeID
            {
                get { return id; }
                set { id = value; }
            }
            public override string ToString()
            {
                return name;
            }
            private readonly List<string> names = new List<string>();
            public string[] Names
            {
                get { return names.ToArray(); }
            }
        }
    }
```

In Listing C.12, values to the UI created in Listing C.11 are added.

8. Add the code, given in Listing C.13, to the Page1.xaml.cs file:

Listing C.13: Adding the Code to the Code-Behind File

In VB

```
    Imports System.Collections.Generic
    Imports System.Linq
    Imports System.Text
    Imports System.Windows
    Imports System.Windows.Controls
    Imports System.Windows.Data
    Imports System.Windows.Documents
    Imports System.Windows.Input
    Imports System.Windows.Media
    Imports System.Windows.Media.Imaging
    Imports System.Windows.Navigation
    Imports System.Windows.Shapes

    Namespace BindingtoCLRObjectVB
        ''' <summary>
        ''' Interaction logic for Page1.xaml
        ''' </summary>
        Partial Public Class Page1
            Inherits Page
            Private name1 As New BindingtoCLRObject.Data()
            Public Sub New()
                InitializeComponent()
                nameGrid.DataContext = name1
```

```
            End Sub
            Private Sub Button_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
                Dim message As String = name1.EmployeeID
                Dim caption As String = name1.Name
                MessageBox.Show(message, caption)
            End Sub
        End Class
    End Namespace
```

In C#

```
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using System.Windows;
    using System.Windows.Controls;
    using System.Windows.Data;
    using System.Windows.Documents;
    using System.Windows.Input;
    using System.Windows.Media;
    using System.Windows.Media.Imaging;
    using System.Windows.Navigation;
    using System.Windows.Shapes;
    namespace BindingtoCLRObjectCS {
        /// <summary>
        /// Interaction logic for Page1.xaml
        /// </summary>
        public partial class Page1 : Page
        {
            private Data name1 = new Data();
            public Page1()
            {
                InitializeComponent();
                nameGrid.DataContext = name1;
            }
            private void Button_Click(object sender, RoutedEventArgs e)
            {
                string message = name1.EmployeeID;
                string caption = name1.Name;
                MessageBox.Show(message, caption);
            }
        }
    }
```

9. Run the BindingtoCLRObject application. The output appears, as shown in Figure C.24:

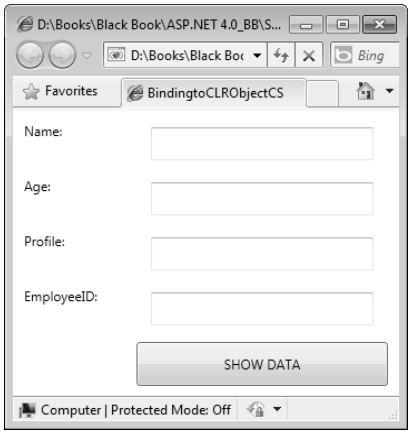

Figure C.24: Displaying the Output of BindingtoCLRObjectCS Application

10. Enter data in the Name, Age, Profile, and EmployeeID text boxes, respectively, and click the SHOW DATA button, as shown in Figure C.25:

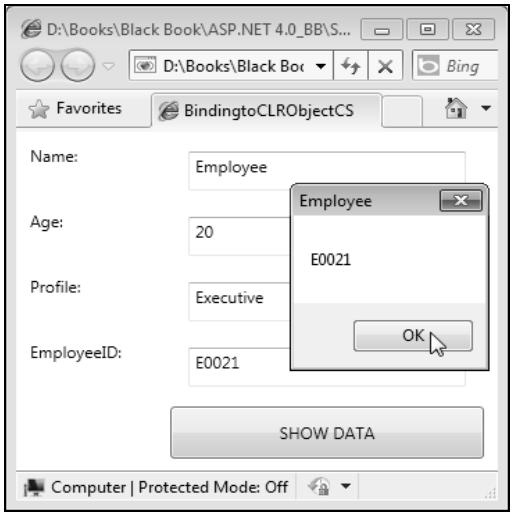

Figure C.25: Displaying Employee Details in a Message Box

As shown in Figure C.25, the employee name and ID that you entered is displayed in the message box.

NOTE

For a WPF browser application to run properly without raising an exception, you first need to set a higher trust level for the WPF browser application and than the default partial trust. To set a higher trust level for your WPF browser application, perform the following steps:

1. *Right-click the name of your WPF browser application in Solution Explorer (Figure C.26).*
2. *Select the Properties option from context menu to open the Properties window of your WPF browser application, as shown in Figure C.26:*

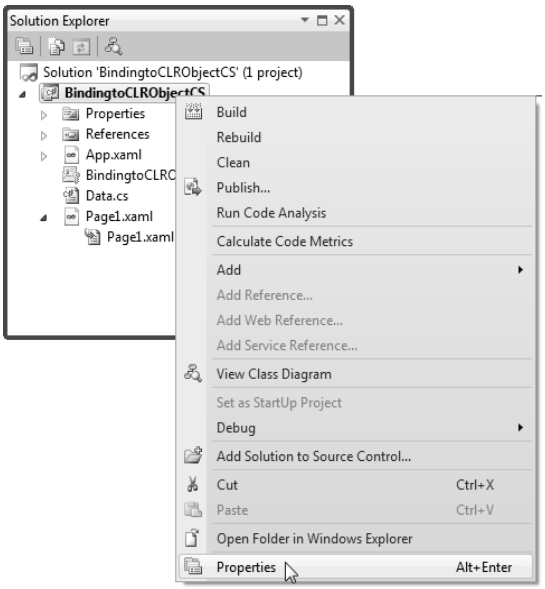

Figure C.26: Opening the Properties Window

3. *Select the Security tab and select the This is a full trust application radio button, as shown in Figure C.27:*

Figure C.27: Modifying the Trust Level of a WPF Browser Application

Next, let's learn how to data bind WPF applications using an ADO.NET object.

ADO.NET

You can also bind an ADO.NET object, such as a DataTable class, to a WPF application. While using an ADO.NET object, you the `IBindingList` interface should be used to provide change notifications. The `IBindingList` interface provides features to support both complex and simple data binding to a data source. While binding your WPF application to an ADO.NET object, the first step is to create a connection. After establishing the connection, an adapter is used which executes SQL statements to retrieve records from a database. The result is then stored in a data table of the data set by calling the Fill method of the adapter. This result is then displayed inside a control in a WPF application.

To bind a WPF application to an ADO.NET object, you first need to create a connection string. You also need to bind controls that you use in your WPF application by using the `ItemSource` and `ItemTemplate` properties. To display a particular column data, you should specify the `DisplayMemberBinding` property.

Now, let's create a WPF browser application named ADO.NETtoWPF in which you can bind an ADO.NET object to a WPF application. You can find the code for this application as ADO.NETtoWPFCS and ADO.NETtoWPFVB in C# and VB, respectively, on the CD. After creating the application, perform the following steps:

1. Add the code, given in Listing C.14, to the Page1.xaml file:

Listing C.14: Preparing the Interface

```
<Page x:Class="ADO.NETtoWPFCS.Page1"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      mc:Ignorable="d"
      d:DesignHeight="300" d:DesignWidth="300"
      Title="Page1">
   <Grid x:Name="Grid1">
       <ListView Name="ListViewEmployeeDetails" Margin="0,0,0,53"
   ItemTemplate="{DynamicResource EmployeeTemplate}" ItemsSource="{Binding Path=Table}"
   HorizontalAlignment="Left" Width="478" Height="172" VerticalAlignment="Bottom">
           <ListView.Background>
```

```
                    <LinearGradientBrush>
                        <GradientStop Color="Bisque"  Offset="0"/>
                    </LinearGradientBrush>
                </ListView.Background>
                <ListView.View>
                    <GridView>
                        <GridViewColumn Header="Employee ID" DisplayMemberBinding="{Binding
                      Path=EmployeeID}"/>
                        <GridViewColumn Header="First Name" DisplayMemberBinding="{Binding
                      Path=FirstName}"/>
                        <GridViewColumn Header="Last Name" DisplayMemberBinding="{Binding
                      Path=LastName}"/>
                        <GridViewColumn Header="City" DisplayMemberBinding="{Binding
                      Path=City}"/>
                        <GridViewColumn Header="Country" DisplayMemberBinding="{Binding
                      Path=Country}"/>
                    </GridView>
                </ListView.View>
            </ListView>
            <Button Height="40" Margin="162,0,163,0" Name="button1" VerticalAlignment="Bottom"
       Click="button1_Click">Button</Button>
            <Label Height="28" HorizontalAlignment="Left" Margin="11,10,0,0" Name="label1"
       VerticalAlignment="Top" Width="142" FontSize="13">Data from SQL Server</Label>
        </Grid>
    </Page>
```

In Listing C.14, the UI on which data is to be displayed is created.

2. Add the code, given in Listing C.15, to the code-behind file:

Listing C.15: Adding the Code to Perform Data Binding

In VB

```
    Imports System.Collections.Generic
    Imports System.Linq
    Imports System.Text
    Imports System.Windows
    Imports System.Windows.Controls
    Imports System.Windows.Data
    Imports System.Windows.Documents
    Imports System.Windows.Input
    Imports System.Windows.Media
    Imports System.Windows.Media.Imaging
    Imports System.Windows.Navigation
    Imports System.Windows.Shapes
    Imports System.Data
    Imports System.Data.SqlClient
    Imports System.Data.Sql

    Namespace ADO.NETtoWPFVB
        ''' <summary>
        ''' Interaction logic for Page1.xaml
        ''' </summary>
        Partial Public Class Page1
            Inherits Page
            Public Sub New()
                InitializeComponent()
            End Sub
            Private Sub button1_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
                Dim mycon As New SqlConnection()
                Dim myadapter As New SqlDataAdapter()
                Dim cmd As New SqlCommand()
                Dim dataquery As [String] = "SELECT EmployeeID, FirstName, LastName,  City,
                     Country FROM Employees"
                cmd.CommandText = dataquery
                myadapter.SelectCommand = cmd
```

```
            mycon.ConnectionString = "Data Source=ANAMIKA-PC\SQLEXPRESS;Initial
                 Catalog=northwind;Integrated Security=True"
            cmd.Connection = mycon
            Dim ds As New DataSet()
            myadapter.Fill(ds)
            ListViewEmployeeDetails.DataContext = ds.Tables(0).DefaultView
            mycon.Close()
        End Sub
    End Class
End Namespace
```

In C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Data;
using System.Data.SqlClient;
using System.Data.Sql;

namespace ADO.NETtoWPFCS
{
    /// <summary>
    /// Interaction logic for Page1.xaml
    /// </summary>
    public partial class Page1 : Page
    {
        public Page1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, RoutedEventArgs e)
         {
                SqlConnection mycon = new SqlConnection();
                SqlDataAdapter myadapter = new SqlDataAdapter();
                SqlCommand cmd = new SqlCommand();
                String dataquery = "SELECT EmployeeID, FirstName, LastName,  City,
                Country FROM Employees";
                cmd.CommandText = dataquery;
                myadapter.SelectCommand = cmd;
                mycon.ConnectionString = "Data Source=ANAMIKA-PC\\SQLEXPRESS;Initial
                Catalog=northwind;Integrated Security=True";
                cmd.Connection = mycon;
                DataSet ds = new DataSet();
                myadapter.Fill(ds);
                ListViewEmployeeDetails.DataContext = ds.Tables[0].DefaultView;
                mycon.Close();
         }
    }
}
```

In Listing C.15, the WPF browser application, ADO.NETtoWPF, is bound to the `Employees` table of the `Northwind` database.

3. Run the ADO.NETtoWPF application. The output appears, as shown in Figure C.28:

Figure C.28: Displaying the Output of ADO.NETtoWPFCS Application

4. Click the Button button to display the data in the list view, as shown in Figure C.29:

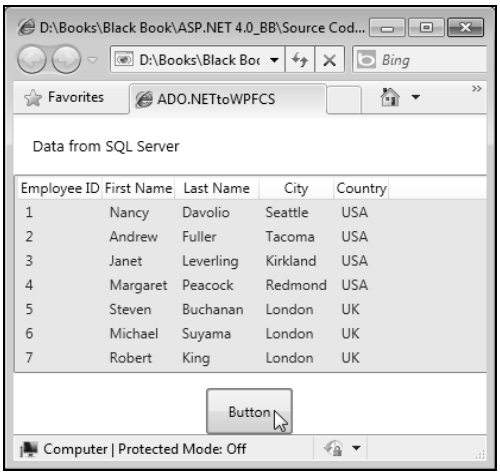

Figure C.29: Displaying the Data

Next, let's learn how to perform data binding with an XML data.

XML

In WPF, you can also bind an XML data to your application by using the `XmlDataProvider` class to access the XML data. The `XmlDataProvider` class helps not only in accessing the data but also displaying the data as a tree of XML nodes. The `XmlDataProvider` class provides a simple and easy way to use any tree of XML nodes as a binding source. When you bind the WPF application to the XML data, you can either include the XML file in your application, or you can create an XML file in the application itself.

Now, let's create a WPF browser application in which you can bind your XML data to the application. You can find the code for this application as BindingXMLObjectCS and BindingXMLObjectVB in C# and VB, respectively, on the CD. After creating the application, you need to first create an XML file. For the XML file, add the code given in Listing C.16 to Notepad and save the file in your project folder with the name `Books.xml`.

Listing C.16: Showing Code for the XML File

```
<?xml version= "1.0" encoding= "utf-8"?>
<Books>
<Book isbn = "100-3-418-1673-1">
<Title>C# 2010 Black Book</Title>j
<Publisher>DreamTech Press</Publisher>
<Author>KogentSolutions Inc.</Author>
</Book>
<Book isbn = "100-4-893-7292-5">
<Title>VB 2010 Black Book</Title>
<Publisher>DreamTech Press</Publisher>
<Author>KogentSolutions Inc.</Author>
</Book>
<Book isbn = "100-7-657-6289-1">
<Title>ASP.NET 4.0 Black Book</Title>
<Publisher>DreamTech Press</Publisher>
<Author>KogentSolutions Inc.</Author>
</Book>
<Book isbn = "187-3-168-9302-1">
<Title>Oracle Black Book</Title>
<Publisher>DreamTech Press</Publisher>
<Author>KogentSolutions Inc.</Author>
</Book>
</Books>
```

After creating the XML file, create a blank WPF browser application named BindingXMLObject and add the code given in Listing C.17 to the Page1.xaml file:

Listing C.17: Binding to an XML File

```
<Page x:Class="BindingXMLObjectCS.Page1"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      mc:Ignorable="d"
      d:DesignHeight="300" d:DesignWidth="300"
      Title="Page1">
    <Page.Resources>
        <XmlDataProvider x:Key="books" Source="Books.xml" XPath="Books"/>
        <DataTemplate x:Key="listTemplate">
            <TextBlock Text="{Binding XPath=Title}"/>
        </DataTemplate>
        <Style x:Key="labelStyle" TargetType="{x:Type Label}">
            <Setter Property="Width" Value="190"/>
            <Setter Property="Height" Value="40"/>
            <Setter Property="Margin" Value="5"/>
            </Style>
    </Page.Resources>
    <Grid DataContext="{Binding Source={StaticResource books}, XPath=Book}">
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition/>
            <RowDefinition/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <ListBox IsSynchronizedWithCurrentItem="True" Margin="5" Grid.Column="0"
          Grid.RowSpan="4" ItemsSource="{Binding}" ItemTemplate="{StaticResource
          listTemplate}"/>
        <Label Style="{StaticResource labelStyle}" Content="{Binding XPath = Title}"
          Grid.Row="0" Grid.Column="1"/>
        <Label Style="{StaticResource labelStyle}" Content="{Binding XPath = Publisher}"
          Grid.Row="1" Grid.Column="1"/>
```

```
            <Label Style="{StaticResource labelStyle}" Content="{Binding XPath = @isbn}"
              Grid.Row="2" Grid.Column="1"/>
        </Grid>

    </Page>
```

In Listing C.17, the data shown in Page1 is the data you have entered in the Books.xml XML file. You can see the final output when you run the application, as shown in Figure C.30:

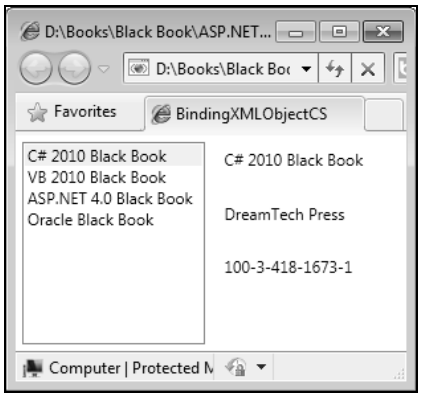

Figure C.30: Displaying Data from the XML File

Next, let's learn how to perform data binding to a dependency object in WPF browser applications.

DependencyObject

You can also bind a WPF browser application to dependency property of any `DependencyObject` class. The major and primary function of the `DependencyObject` class in a WPF application is to compute the value of properties and provide the system notification about the values that have changed. The `DependencyProperty.DependencyProperty` class enables the registration of dependency property into the property system. It also provides identification and information about each dependency property. You can register the dependency property in a WPF application by calling the Register method.

Now, let's create a WPF browser application named DependenecyObjectWPF. You can find the code for this application as DependenecyObjectWPFCS and DependenecyObjectWPFVB in C# and VB, respectively. After creating the application, add the code given in Listing C.18 to the Page1.xaml file:

Listing C.18: Implementing DependencyObject Class

```
    <Page x:Class="DependenecyObjectWPFCS.Page1"
          xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
          xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
          xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
          mc:Ignorable="d"
          d:DesignHeight="300" d:DesignWidth="300"
          Title="Page1">
        <Grid>
            <Label Height="28" Margin="64,30,94,0" Name="label1" VerticalAlignment="Top"
              FontSize="15" FontWeight="Bold">Choose a Color:</Label>
            <ComboBox Height="23" Margin="47,65,53,0" Name="comboBox1"
              VerticalAlignment="Top">
                <ComboBoxItem>RED
                </ComboBoxItem>
                <ComboBoxItem>BLUE
                </ComboBoxItem>
                <ComboBoxItem>YELLOW
                </ComboBoxItem>
                <ComboBoxItem>BLACK
                </ComboBoxItem>
            </ComboBox>
```

```
            <Canvas Margin="89,0,89,2" Height="100" VerticalAlignment="Bottom">
                <Canvas.Background>
                    <Binding ElementName="comboBox1" Path="SelectedItem.Content"/>
                </Canvas.Background>
            </Canvas>
        </Grid>
    </Page>
```

In Listing C.18, binding is performed on a `ComboBox` control, combobox1 and a `Canvas` control. In this example, as you select a particular color in the combobox1 control, the background of the Canvas control changes to the selected color. This is done using the `DependencyObject` class. The `Background` property of the Canvas control is a dependency property.

When you run the DependenecyObjectWPF application, the output appears, as shown in Figure C.31:

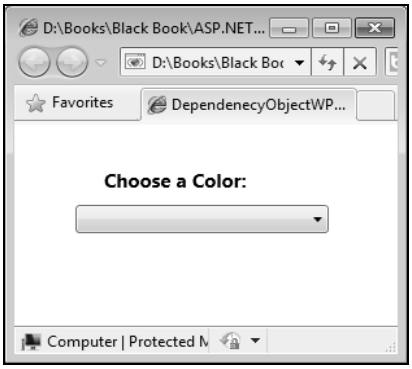

Figure C.31: Displaying the Output of the DependencyObjectWPF Application

If you click the Choose a Color combo box and select a color, the final output appears, as shown in Figure C.32:

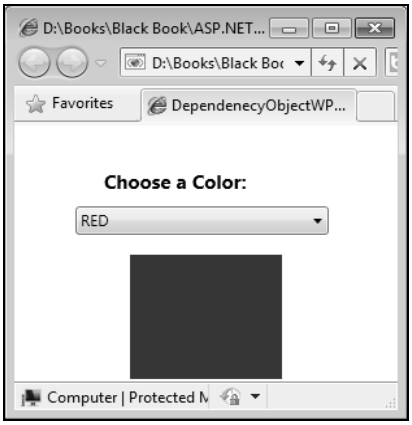

Figure C.32: Displaying the Output after Selecting a Color

Next, let's learn about the different ways of declaring data binding in WPF.

Declaration of Data Binding

You can implement binding in WPF in different ways and format. You can implement binding in the XAML format, which is markup file of a WPF application. You can also implement binding through the code-behind file of a WPF application. The third way to create binding in WPF is to specify the Path property in the XAML code. You can specify the source value which you want to bind by using the Path property. Now, let's see how you can create data binding in a WPF application in these three ways.

XAML Based Binding

You can bind a WPF application in the XAML format by using the Binding property, which is the markup extension. When you use Binding property as an extension to declare binding, the declaration consists a series of clauses. The clauses are in the `Name = Value` pair, where `Name` is the name of the `Binding` property and `Value` is the value you are setting for the property. You should note that when you create binding in the XAML format, the Binding property must be attached to a specific dependency property of the target object. The following code snippet shows the basic syntax of using the `Binding` property in the XAML format:

```
<TextBox Text="{Binding Source={StaticResource myDataSource}, Path=Name}"/>
```

In the preceding code snippet, the `Text` property of the `TextBox` control is using the `Binding` property.

Code Based Binding

Another way to implement binding is to set the properties directly on a `Binding` object in code. The `FrameworkElement` and `FrameworkContentElement` classes provide the `SetBinding` method to bind a control to a data in code. The following code snippet shows how to bind your WPF application in code:

```
BindingOperations.SetBinding(myDateText, TextBlock.TextProperty, myNewBindDef);
```

In the preceding code snippet, the `Text` property of the `TextBlock` control is bound to data using the SetBinding() method.

Binding Path Syntax

You can also use the `Path` property to specify the source value which you want to bind in a WPF application. The `Path` property specifies the source object used for binding. For example, you can bind the `Text` property of a `TextBox` control. You can also bind an attached property of a control using the `Path` property. For instance, to bind the attached property `DockPanel.Dock`, the syntax that you can use with the `Path` property is as follows:

```
Path = (DockPanel.Dock)
```

You can also bind a property of a control to a particular field of a database using the `Path` property. The following code snippet shows how to bind a property of a control to a particular field of a database using the Path property:

```
Path=Employees.FirstName
```

In the preceding code snippet, a control is bound to the `FirstName` column of the `Employees` table.

Next, let's learn how to perform data binding with Web services in WPF applications.

Data binding to Web Services in WPF

You can also bind your WPF applications to a Web service by first retrieving data in a WPF application and then adding a Web service to your WPF application. Let' create an application in which you can bind your WPF application to a Web service by performing the following steps:

1. Open VS 2010 and select File → New → Project from the menu bar. The New Project dialog box opens.
2. Select the ASP.NET Empty Web Application option from the middle pane of the New Project dialog box.
3. Enter the name DataServiceWebApplication in the Name text box and specify an appropriate location to save the application in the Location combo box. You can find the code for this application as DataServiceWebApplicationCS and DataServiceWebApplicationVB in C# and VB, respectively, on the CD.
4. Right-click the name of your application, DataServiceWebApplication, in Solution Explorer and select Add→New Item from the context menu.
5. Select the Web Service option from the middle pane of the Add New Item dialog box and name it as DataBindingWebService.asmx.
6. Click the Add button to add the Web service, DataBindingWebService.asmx, to the DataServiceWebApplication application.
7. Add the code, given in Listing C.19, to the code-behind file of the Web service, DataBindingWebService.asmx:

Listing C.19: Creating a Web Service

In VB

```
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.ComponentModel
Imports System.Data
Imports System.Data.SqlClient

' To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the
  following line.
' <System.Web.Script.Services.ScriptService()> _
<System.Web.Services.WebService(Namespace:="http://tempuri.org/")> _
<System.Web.Services.WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<ToolboxItem(False)> _
Public Class DataBindingWebService
    Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function GetData() As DataSet
        Dim sqlConnection1 As New System.Data.SqlClient.SqlConnection()
        sqlConnection1.ConnectionString = "Data Source=ANAMIKA-PC\SQLEXPRESS;Initial
         Catalog=northwind;Integrated Security=True"
        Dim selectStr As String = "SELECT EmployeeID, FirstName, LastName,  City, Country
         FROM Employees"
        Dim da As New SqlDataAdapter(selectStr, sqlConnection1)
        Dim ds As New DataSet()
        da.Fill(ds)
        Return (ds)
    End Function

End Class
```

In C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Data;
using System.Data.SqlClient;

namespace DataServiceWebApplication
{
    /// <summary>
    /// Summary description for DataBindingWebServiceCS
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    // To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment
         the following line.
    // [System.Web.Script.Services.ScriptService]
    public class DataBindingWebServiceCS : System.Web.Services.WebService
    {

        [WebMethod]
        public DataSet GetData()
         {
                System.Data.SqlClient.SqlConnection sqlConnection1 = new
                 System.Data.SqlClient.SqlConnection();
                sqlConnection1.ConnectionString = "Data Source=ANAMIKA-
                PC\\SQLEXPRESS;Initial Catalog=northwind;Integrated Security=True";
                string selectStr = "SELECT EmployeeID, FirstName, LastName,  City,
                Country FROM Employees";
```

```
                    SqlDataAdapter da = new SqlDataAdapter(selectStr, sqlConnection1);
                    DataSet ds = new DataSet();
                    da.Fill(ds);
                    return (ds);
                }
            }
        }
```

Listing C.19 creates a Web service, which retrieves data in a WPF application. You can see the output of the Web service (by pressing the F5 key) in Figure C.33:

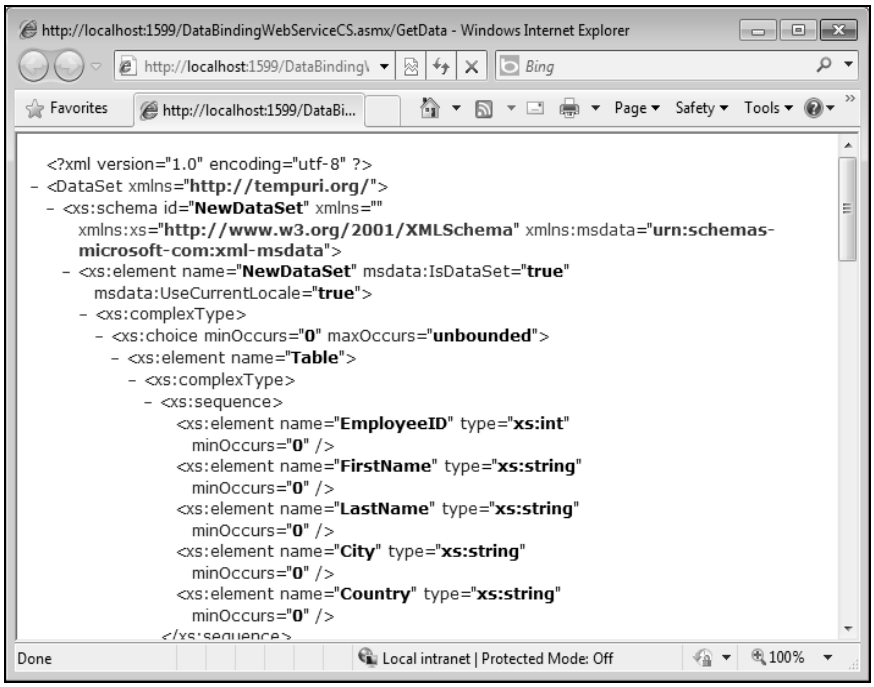

Figure C.33: Displaying the Output of the Web Service

8. Right-click the solution name, DataServiceWebApplication in Solution Explorer and select Add→New Project from the context menu.
9. Select WPF Browser Application from the middle pane of the Add New Project dialog box and name it as MyWpf.
10. Click the OK button to add the MyWpf application to your DataServiceWebApplication application.
11. Add the code, given in Listing C.20, to the Page1.xaml file.

Listing C.20: Creating the Interface

```
<Page x:Class="MyWpf.Page1"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      mc:Ignorable="d"
      d:DesignHeight="300" d:DesignWidth="300"
      Title="Page1">
    <Grid x:Name="Grid1">
        <ListView Name="ListViewEmployeeDetails" Margin="61,70,39,120"
  ItemTemplate="{DynamicResource EmployeeTemplate}" ItemsSource="{Binding Path=Table}">
            <ListView.Background>
                <LinearGradientBrush>
                    <GradientStop Color="Bisque"  Offset="0"/>
```

```
                </LinearGradientBrush>
            </ListView.Background>
            <ListView.View>
                <GridView>
                    <GridViewColumn Header="Employee ID" DisplayMemberBinding="{Binding
                        Path=EmployeeID}"/>
                    <GridViewColumn Header="First Name" DisplayMemberBinding="{Binding
                        Path=FirstName}"/>
                    <GridViewColumn Header="Last Name" DisplayMemberBinding="{Binding
                        Path=LastName}"/>
                    <GridViewColumn Header="City" DisplayMemberBinding="{Binding
                        Path=City}"/>
                    <GridViewColumn Header="Country" DisplayMemberBinding="{Binding
                        Path=Country}"/>
                </GridView>
            </ListView.View>
        </ListView>
        <Button Height="40" Margin="161,0,164,53" Name="button1"
         VerticalAlignment="Bottom"
    Click="button1_Click">Button</Button>
        <Label Height="28" HorizontalAlignment="Left" Margin="11,10,0,0" Name="label1"
    VerticalAlignment="Top" Width="142" FontSize="13">Data from Web Service</Label>
     </Grid>
</Page>
```

12. Add a reference of the Web service, DataBindingWebService.asmx to the WPF application by performing the following steps:
13. Right-click the MyWpf application in Solution Explorer and select Add Service Reference from the context menu. The Add Service Reference dialog box appears.
14. Click the Discover button in the Add Service Reference dialog box. This adds the available Web services to the Services pane in the Add Service Reference dialog box, as shown in Figure C. 34:

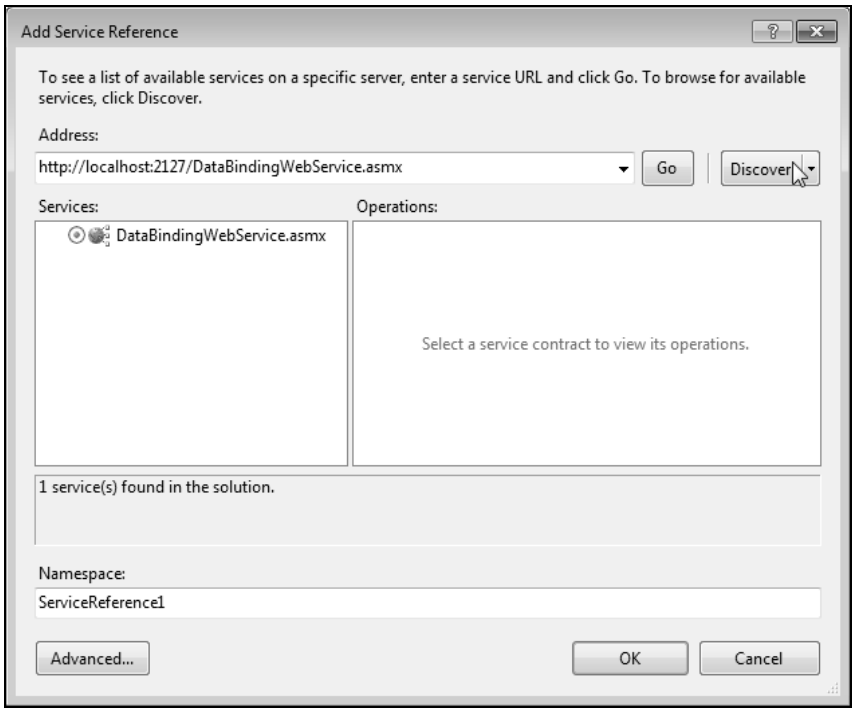

Figure C.34: Locating a Web Service

15. Select the Web service, DataBindingWebService, listed in the Services pane and click the OK button, as shown in Figure C.35:

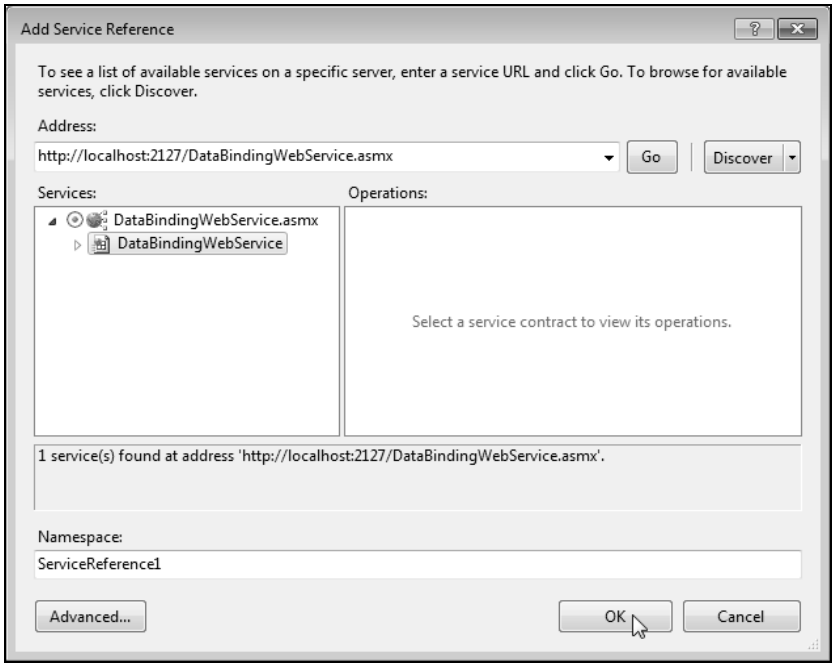

Figure C.35: Adding a Web Service

16. Build the solution by pressing the F6 key and add the code given in Listing C.21 to the code-behind file of the MyWpf application:

Listing C.21: Adding the Code to the MyWpf Application

In VB

```
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Input
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Navigation
Imports System.Windows.Shapes
Imports System.ComponentModel
Imports MyWpf.ServiceReference1

Namespace MyWpf
    ''' <summary>
    ''' Interaction logic for Page1.xaml
    ''' </summary>

    Partial Public Class Page1
        Inherits Page
        Public Sub New()
            InitializeComponent()
        End Sub
```

```
            Private Sub button1_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
                Dim s As DataBindingWebServiceSoapClient = New
                     ServiceReference1.DataBindingWebServiceSoapClient()
                ListViewEmployeeDetails.DataContext = s.GetData()
            End Sub

        End Class
    End Namespace
```

In C#

```
    using System;
    using System.Collections.Generic;

    using System.Linq;
    using System.Text;

    using System.Windows;
    using System.Windows.Controls;

    using System.Windows.Data;
    using System.Windows.Documents;

    using System.Windows.Input;
    using System.Windows.Media;
    using System.Windows.Media.Imaging;

    using System.Windows.Navigation;
    using System.Windows.Shapes;

    using System.ComponentModel;
    using MyWpf.ServiceReference1;

    namespace MyWpf

    {
        /// <summary>
        /// Interaction logic for Page1.xaml
        /// </summary>
        public partial class Page1 : Page
        {
            public Page1()
            {
                InitializeComponent();

            }

            private void button1_Click(object sender, RoutedEventArgs e)
            {
                DataBindingWebServiceCSSoapClient s = new
                     ServiceReference1.DataBindingWebServiceCSSoapClient();
                ListViewEmployeeDetails.DataContext=s.GetData();
            }

        }
    }
```

In Listing C.21, the code is added on the click event of button, button1, to generate the data, which is retrieved by using the Web service, DataBindingWebService. You can see the final output in Figure C.36:

Figure C.36: Displaying Data Retrieved from a Web Service

NOTE

To see the output, you need to set your WPF browser application, MyWpf, as the start-up project. Right-click MyWpf in Solution Explorer and select the Set as StartUp Project option from the context menu and then run your application.

With this, we come to the end of this appendix wherein we have learned about WPF and creating XBAP applications.