22

Rich ASP.NET Controls

OBJECTIVES

- Choose from multiple items with the RadioButtonList and CheckBoxList controls
- Select dates using the Calendar control
- Display randomly selected images using the AdRotator control
- Insert HTML into a page using the Literal control
- Add controls at runtime using the PlaceHolder control

Throughout this book, you've worked with many different ASP.NET server controls, including the Label, TextBox, DropDownList, ListBox, and Button controls. Beyond the controls that map directly to HTML controls, ASP.NET offers a series of rich controls that provide functionality HTML controls simply can't provide.

In this chapter, you'll investigate several additional ASP.NET server controls, including the following:

- The CheckBoxList and RadioButtonList controls
- The Calendar control
- The AdRotator control
- The Literal control
- The PlaceHolder control

You can find a separate, prebuilt project that demonstrates these controls with the examples for this book in the `MiscWebControls` folder. Load the `MiscWebControls.sln` solution file, and you'll be able to follow along with the discussion in this chapter.

NOTE

We can't even begin to attempt to cover every detail of each of these controls in this chapter. We'll merely demonstrate the behavior of each control and how you might use it, and we'll leave the "heavy lifting" to you.

The CheckBoxList and RadioButtonList Controls

ASP.NET's RadioButton and CheckBox controls work fine, but they really provide no support for handling groups of related data. If you want to fill a list of either type of controls with data from a DataSet, for example, you'll need to fill each control's text individually. In addition, keeping track of which items in the list of individual controls have been selected is a chore.

To make working with groups of these controls easier, ASP.NET provides the CheckBoxList and RadioButtonList controls. Both controls allow you to bind their lists to data sources and fill these lists at runtime. The major difference between the two is that you can select as many items as you require using the CheckBoxList control, but you can only select a single item using the RadioButtonList control. (This behavior is most likely what you'd expect, given the usage of these two types of input controls.)

Imagine that you'd like to allow users to select a single region from a list of regions. You might like to allow a single selection (using a list of radio buttons) or multiple selections (using check boxes). The sample page, `ListControls.aspx`, provides this capability using both types of list controls (see Figure 22.1).

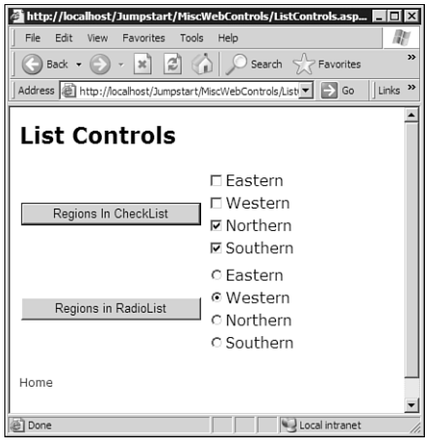

FIGURE 22.1 Use the CheckBoxList and RadioButtonList controls to allow selections from a list of options.

Each of the two buttons on `ListControls.aspx` calls a common procedure: `RegionLoad`. This procedure fills the list with data retrieved from the Regions table in the Northwind sample database.

```
Private Sub btnRegionCheckList_Click( _
 ByVal sender As System.Object, _
```

```
 ByVal e As System.EventArgs) _
 Handles btnRegionCheckList.Click

  RegionLoad(clstRegions)
End Sub

Private Sub btnRegionRadioList_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) _
 Handles btnRegionRadioList.Click

  RegionLoad(rlstRegions)
End Sub
```

The `RegionLoad` procedure accepts, as its parameter, a variable of the `ListControl` type. Because both the CheckBoxList and RadioButtonList controls inherit from the `ListControl` base class, either is a valid parameter for this procedure. (You could also pass in a ListBox control or DropDownList control, because those controls also inherit from the `ListControl` base class.)

The `RegionLoad` procedure shown in Listing 22.1 uses code exactly like you've seen before for working with ListBox or DropDownList controls.

LISTING 22.1 Fill Lists with Region Information Using This Procedure

```
Private Sub RegionLoad( _
 ByVal ctlRegions As ListControl)

  Dim strSQL As String
  Dim strConn As String
  Dim ds As DataSet

  ' Build Connect and SQL strings
  strConn = "Provider=sqloledb;" & _
   "Data Source=(local);" & _
   "Initial Catalog=Northwind;User ID=sa"
  strSQL = _
   "SELECT RegionID, RegionDescription " & _
   "FROM Region"

  ds = GetDataSet(strSQL, strConn)

  With ctlRegions
```

LISTING 22.1 Continued

```
    .DataTextField = "RegionDescription"
    .DataValueField = "RegionID"
    .DataSource = ds
    .DataBind()
  End With
End Sub
```

This procedure builds SQL and connection strings and then calls the `GetDataSet` procedure to retrieve a `DataSet` object. The code sets the `DataTextField` and `DataValueField` properties of the list control, sets the data source to the DataSet, and then calls the `DataBind` method to hook up the data and display the control.

The `GetDataSet` procedure should look completely familiar to you by this point. This procedure accepts SQL and connection strings and returns a `DataSet` object:

```
Private Function GetDataSet( _
 ByVal SQL As String, _
 ByVal ConnectionString As String) _
 As DataSet

  Dim da As OleDbDataAdapter
  Dim ds As New DataSet()

  Try
    da = New OleDbDataAdapter(SQL, ConnectionString)
    da.Fill(ds)

  Catch
    Throw

  End Try
  Return ds
End Function
```

The Calendar Control

If you want to allow a user to select a date, displaying a calendar is the right way to do it. You could fashion your own calendar using an HTML table and a lot of scripting code (many developers have done this, in many different ways), but it's a lot of effort. Fortunately, ASP.NET handles this effort for you. The Calendar control, shown

in Figure 22.2 (see the sample page `CalendarControl.aspx`) displays a neatly formatted, bindable table-like view that provides for almost any "look" you need. (Actually, if you view the source for the page containing the Calendar control in the browser, you'll see that it actually is an HTML table, albeit a somewhat complex one.)

FIGURE 22.2 Use the Calendar control to select a date.

The Calendar control uses client-side script to trigger a postback to the server each time you click any of the dates, or other links, on the calendar. As you can see, in the status bar in Figure 22.2, each date calls a JavaScript procedure that raises an event on the server, thus allowing you to run code each time the user clicks a date.

When the user clicks a date, the `SelectionChanged` event is triggered. In this event procedure, you might retrieve the `SelectedDate` property of the Calendar control and use that date:

```
Private Sub cal_SelectionChanged( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) _
 Handles cal.SelectionChanged

  lblDate.Text = _
   String.Format("Selected date: {0:d}", _
   cal.SelectedDate)
End Sub
```

TIP

This example uses a format specifier in the `String.Format` placeholder: `{0:d}`. The formatting string, `d`, indicates that the date should be formatted using the short date format of the current locale. For more information on formatting strings, see the online help for the `String.Format` method.

If you click one of the links at the top of the control that moves you to a new month, you trigger the Calendar control's `VisibleMonthChanged` event. This event procedure receives, in its second parameter, a variable of the `MonthChangedEventArgs` type. This variable provides two unique properties, `NewDate` and `PreviousDate`, that allow you to determine the first day of the month that's currently showing and the first day of the month that was previously showing. The sample page uses the `NewDate` property to display the current month:

```
Private Sub cal_VisibleMonthChanged( _
 ByVal sender As System.Object, _
 ByVal e As System.Web.UI.WebControls. _
 MonthChangedEventArgs) _
 Handles cal.VisibleMonthChanged

  lblMonth.Text = _
   String.Format("Visible Month: {0:MMMM yyyy}", _
   e.NewDate)
End Sub
```

TIP

This example uses a user-defined formatting specification, `MMMM yyyy`, to display only the month name and the year. Again, see the online help for `String.Format` for complete information on its formatting capabilities.

Initializing the Calendar

The Calendar control provides two properties, `SelectedDate` and `TodaysDate`, that are closely related. The `SelectedDate` property (and its cousin, `SelectedDates`) allows you to specify or retrieve the selected date—that is, the date the user has selected. (The `SelectedDates` property allows you to set or retrieve a group of selected dates.) The `TodaysDate` property allows you to set or retrieve the date the calendar considers to be the current date, and this property will contain the server's date if you don't specify a date.

It's quite possible, and highly probable, that these two properties will contain different values. When you first load a page containing the Calendar control, however,

you might like them to both contain the same value. (If you don't specify a value for the `SelectedDate` property, it contains the date 1/1/0001. If you don't specify a value for the `TodaysDate` property, it contains the current date on the server.)

To avoid any confusion that might occur if you want to allow a user to select the current date without actually clicking anything, you might want to initialize both these properties to the same date as your page loads:

```
Private Sub Page_Load( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) _
 Handles MyBase.Load

  cal.SelectedDate = cal.TodaysDate
End Sub
```

The AdRotator Control

The AdRotator control allows you to display a randomly selected image, selected from a list contained within an XML file, each time your page is rendered. You supply the images and the XML file containing information about the images, and the ASP.NET page framework handles the rest. You can apply filters dynamically so that you can limit the images to subsets of your choosing, as well. In the sample page, `AdRotatorControl.aspx`, you can select to show images of either male or female employees (or both) randomly selected from a list of images in an XML file (see Figure 22.3).

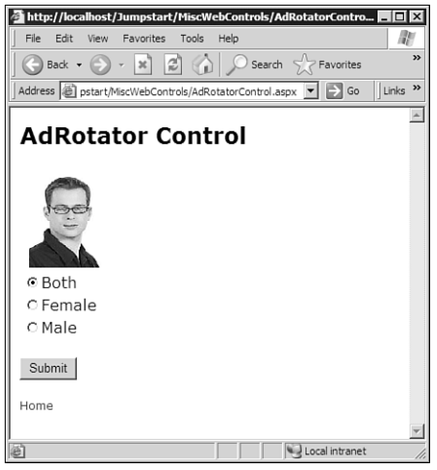

FIGURE 22.3 The AdRotator control allows you to cycle through a series of images.

To use the AdRotator control, you place the control on a page and set the control's `AdvertisementFile` property to the location of the XML file containing the image information. In addition, you can set the `KeywordFilter` property of the AdRotator control at any time to filter the images based on the keyword you specify.

The sample advertisement file, `Ads.xml`, contains information about each of the images to be displayed, like this:

```
<Advertisements>
<Ad>
  <ImageUrl>images/buchanan.jpg</ImageUrl>
  <NavigateUrl>
    ClickThrough.aspx?Name=Buchanan
  </NavigateUrl>
  <AlternateText>Buchanan</AlternateText>
  <Impressions>80</Impressions>
  <Keyword>Male</Keyword>
</Ad>
<Ad>
  <ImageUrl>images/callahan.jpg</ImageUrl>
  <NavigateUrl>
    ClickThrough.aspx?Name=Callahan
  </NavigateUrl>
  <AlternateText>Callahan</AlternateText>
  <Impressions>80</Impressions>
  <Keyword>Female</Keyword>
</Ad>
</Advertisements>
```

The `Advertisements` and `Ad` tags are required, and Table 22.1 describes each of the elements within each ad.

TABLE 22.1 Use These Elements Within Your Advertisements File

Attribute	Description
`ImageUrl`	The URL of the image to display.
`NavigateUrl`	The URL of the page to navigate to when the AdRotator control is clicked.
`AlternateText`	The text to display if the image is unavailable. On some browsers, this text is displayed as a tooltip.
`Impressions`	A value that indicates how often an advertisement is displayed in relation to other advertisements in the XML file.

TABLE 22.1 Continued

Attribute	Description
`Keyword`	The category for the advertisement. This is used by the AdRotator control to filter the list of advertisements for a specific category.

TIP

The `Impressions` element is somewhat confusing. The values you enter here are relative. That is, you determine the scale for the meaning of these values. If all images have equal values for this element, they're all equally likely to appear. If you have five images, and the `Impressions` values are 5, 4, 3, 2, and 1, then the first image is five times more likely to appear than the last. In other words, out of 15 hits, the first image is likely to appear five times, the second four times, and so on. You would get the same behavior using values such as 100, 80, 60, 40, and 20. The magnitude of these values doesn't matter, only the relative weights.

In this example, clicking the AdRotator control takes you to `ClickThrough.aspx`, which displays the information passed to the page, using the `Request.QueryString` method (see the `NavigateURL` elements in the sample XML):

```
Private Sub Page_Load( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles MyBase.Load

  If Not Page.IsPostBack Then
    If Not IsNothing(Request.QueryString("Name")) Then
      lblInfo.Text = "You clicked on " & _
       Request.QueryString("Name")
    End If
  End If
End Sub
```

Filtering the AdRotator Control

If you supply `Keyword` elements in the XML advertisements file, you can filter the images displayed in the AdRotator control. Set the `KeywordFilter` property of the control and only see images whose keyword matches the keyword you've specified.

In the sample advertisements file, all the images contain either "Male" or "Female" as their keyword values. The sample page allows you to specify whether you want

male or female images, or both. (Specifying "Both" on the sample page sets the `KeywordFilter` property to an empty string, effectively allowing all images.)

```
Private Sub rblFilter_SelectedIndexChanged( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) _
 Handles rblFilter.SelectedIndexChanged

  Dim strFilter As String
  strFilter = rblFilter.SelectedItem.Text
  If strFilter = "Both" Then
    strFilter = String.Empty
  End If
  adEmp.KeywordFilter = strFilter
End Sub
```

The Literal Control

When you start creating data-driven ASP.NET sites, you'll often need to dynamically generate HTML content for your pages. Perhaps you need to store item description information, as HTML, in a database table. At runtime, you need to be able to display that content, rendered correctly as HTML. The Literal control can render HTML at runtime, allowing you to inject HTML into a page.

The sample page, `LiteralControl.aspx`, allows you to enter HTML into a TextBox control. When you click Display in Literal Control, you execute this event procedure, copying the text into a Literal control:

```
Private Sub btnAssign_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) _
 Handles btnAssign.Click

  litHTML.Text = txtHTML.Text
End Sub
```

That's all it takes to insert HTML into a page at runtime. (Imagine that the HTML was coming from a table, rather than from a TextBox control. Things get awfully powerful, awfully quickly. You can render an entire page from a database, modifying the layout by changing the data in a table. The possibilities are huge.) Figure 22.4 demonstrates the technique, rendering a bulleted list from HTML entered into the text box on the page.

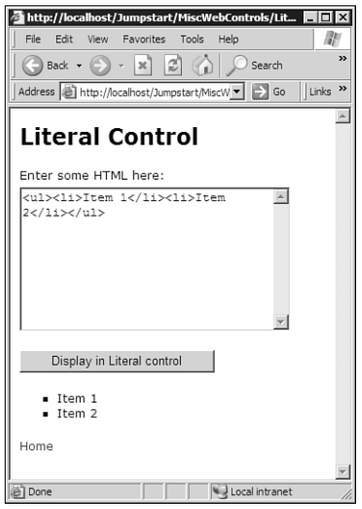

FIGURE 22.4 Use the Literal control to inject HTML into a page. The bulleted list on this page gets its HTML from the TextBox control at the top.

The PlaceHolder Control

Just as the Literal control allows you to inject HTML into a page at runtime, the PlaceHolder control allows you to inject new controls into a page, at a specific location, at runtime. Imagine that you have a table full of links, and you want to insert the series of links into a page between to static paragraphs of text. You don't want to render the entire page at runtime, only the list of links.

No problem—the PlaceHolder control comes to the rescue (working with its pal, the Literal control). We've used this technique to generate the content for `Main.aspx` in the sample project shown in Figure 22.5. At design time, there's really nothing on this page except a Label control and a single PlaceHolder control (see Figure 22.6).

The PlaceHolder control provides a collection property, the `Controls` property, that allows you to work with and add to the collection of controls contained within the PlaceHolder control. (The PlaceHolder control isn't the only control that provides a `Controls` collection—the GroupBox, RadioButtonList, and CheckBoxList controls, for example, also allow you to work with the subcontrols contained within the control. The PlaceHolder control, however, is the best control to use if you want to generate new content on a page at runtime.)

FIGURE 22.5 You can generate any control at runtime and use the PlaceHolder control to place the new control at a known location on the page.

FIGURE 22.6 The sample page doesn't contain much until runtime.

To add a new control at runtime, you can simply instantiate the type of control you want, set the properties you need, and then add the new control to the `Controls` collection of the PlaceHolder control. For example, the `Main.aspx` page needs four HyperLink controls, with <p> elements (paragraph breaks) between them. The page contains the `AddLink` procedure, shown in Listing 22.2, which adds a single link and a <p> element afterwards:

LISTING 22.2 Add HyperLink Controls to the Page Dynamically

```
Private Sub AddLink( _
 ByVal NavigateURL As String, _
 ByVal Text As String, _
 ByVal ID As String)

  Dim hyp As New HyperLink()
  Dim lit As New Literal()

  With hyp
    .NavigateUrl = NavigateURL
```

LISTING 22.2 Continued

```
    .Text = Text
    .ID = ID
  End With
  plcHyper.Controls.Add(hyp)

  lit.Text = "<p>"
  plcHyper.Controls.Add(lit)
End Sub
```

The `AddLink` procedure starts out by instantiating a new HyperLink control and a new Literal control:

```
Dim hyp As New HyperLink()
Dim lit As New Literal()
```

The procedure sets properties of the HyperLink control using parameters passed into the procedure:

```
With hyp
  .NavigateUrl = NavigateURL
  .Text = Text
  .ID = ID
End With
```

Once the hyperlink is all set up, the code adds the control to the `Controls` collection of the PlaceHolder control on the page:

```
plcHyper.Controls.Add(hyp)
```

The code then sets the `Text` property of the Literal control and adds it to the PlaceHolder, as well:

```
lit.Text = "<p>"
plcHyper.Controls.Add(lit)
```

The `Page_Load` procedure calls the `AddLink` procedure four times, once for each link it must add:

```
Private Sub Page_Load( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) _
 Handles MyBase.Load
```

```
  If Not Page.IsPostBack Then
    AddLink("ListControls.aspx", _
     "List Controls", "hypListControls")
    AddLink("CalendarControl.aspx", _
     "Calendar Control", "hypCalendarControl")
    AddLink("AdRotatorControl.aspx", _
     "Ad Rotator Control", "hypAdRotator")
    AddLink("LiteralControl.aspx", _
     "Literal Control", "hypLiteralControl")
  End If
End Sub
```

Although the sample page uses hard-coded links, your page might use link information pulled from a database, thus allowing you to create totally dynamic pages at runtime.

Summary

Although many of the ASP.NET controls map directly to existing HTML controls, the controls shown in this chapter don't. Each of the controls shown here adds its own utility:

- The Calendar control aggregates existing HTML controls.
- The Literal and Placeholder controls allow you to create dynamic content.

You may not use these controls in every page you build, but it's useful to know that these extended controls are available, should you ever need them.