

3

Custom Controls

3.0. Introduction

One of the most powerful features of ASP.NET is its support for custom server controls and components. ASP.NET ships with dozens of built-in controls, and developers can easily extend these controls or write their own controls from scratch. Server controls can be used to encapsulate complex user interface logic or business rules, and can benefit from design-time support like drag-and-drop and toolbox support and property builders. Custom controls pick up where User Controls leave off, providing greater flexibility, reusability, and a better design time experience, but with some added complexity. In this chapter you will find examples covering some of the most common server-control techniques.

Custom controls as a rule inherit directly or indirectly from the `System.Web.UI.Control` base class. Controls that are visible on a page should inherit directly or indirectly from `System.Web.UI.WebControls.WebControl`, which provides properties like `style`, which you can use to determine the look of the control on the page. Custom controls can be built in a number of ways. Some simply override the `Render()` method, thus determining the HTML output in place of the control at runtime. Others, known as composite controls, act as containers for other controls. Others inherit from existing fully functional controls to create more specific versions of these controls or to enhance their functionality.

Controls in ASP.NET can support *data-binding* as well as *templates*. In fact, there is easily enough information about building controls to fill an entire book (and in fact such a book exists and is listed at the end of this section), so this chapter attempts to cover the most common techniques that you will use, leaving much of the theory to other books dedicated to control building.

See Also

Developing ASP.NET Server Controls and Components by Nikhil Kothari and Vandana Datye (Microsoft Press; ISBN 0735615829)

3.1. Declaring a Simple Custom Control

You want to create a simple custom control to output some text.

Technique

This example demonstrates how easy it is to create custom controls in ASP.NET, especially when compared to COM components. You simply create a class that inherits from `System.Web.UI.Control` or `System.Web.UI.WebControls.WebControl` and give it whatever properties and methods you need. In Visual Studio .NET, you would normally do this by creating a new Web Control Library project. You override its `Render()` method to control its output, and you have a very simple yet powerful tool for encapsulating and reusing user interface logic.

The `Recipe0301vb` class is as follows:

```
Imports System.ComponentModel
Imports System.Web.UI
Namespace AspNetCookbook

    <DefaultProperty("Text"), ToolboxData("<{0}:Recipe0301vb
runat=server></{0}:Recipe0301vb>")> Public Class Recipe0301vb
    Inherits System.Web.UI.WebControls.WebControl

        Dim _text As String

        <Bindable(True), Category("Appearance"),
        DefaultValue("")> Property [Text]() As String
            Get
                Return _text
            End Get

            Set(ByVal Value As String)
                _text = Value
            End Set
        End Property

        Protected Overrides Sub Render(
            ByVal output As System.Web.UI.HtmlTextWriter)
            output.Write([Text])
        End Sub

    End Class
End Namespace
```

To reference a custom control on a Web Form, you need to add a `Register` directive to the page, and specify three parameters. The `TagPrefix` is used for all controls from this

namespace and assembly when they are declared on the page, and can be anything but `asp`, which is reserved for the built-in Web Controls that ASP.NET provides. Next, the namespace in which the controls reside must be specified. Finally, the name of the assembly, without any path information or the `.DLL` extension, is specified for the `Assembly` parameter. An example of this follows:

```
<%@ Page language="VB" %>
<%@ Register TagPrefix="AspNetCookbook" Namespace="AspNetCookbook"
Assembly="RecipesVB" %>
...
<form id="Form1" method="post" runat="server">
  <AspNetCookbook:Recipe0301vb id="Recipe0301vb1" runat="server" />
</form>
```

Comments

Note that in Visual Studio .NET, a default namespace with the same name as the project is automatically prefixed to all Visual Basic class names. This is a frequent source of confusion and is inconsistent with how default namespaces are handled in C#, where they are inserted into each class file as a visible namespace. You can turn off this default behavior by setting the default namespace to an empty string in the Project Properties dialog box. You can determine the full namespace of a class by using the `ILDASM.EXE` command-line tool on the generated assembly, or by going into the class view utility in Visual Studio .NET.

3.2. Extending Existing Web Controls

You want to extend the functionality of an existing Web Control.

Technique

This example shows how to extend the functionality of an existing control—the `Label` control—and turn it into a `RainbowLabel`. This is accomplished through the use of *inheritance*—`RainbowLabel` is a *subclass* of the `System.Web.UI.WebControls.Label` class.

The `RainbowLabel` class is as follows:

```
Imports System
Imports System.Web.UI
Imports System.Web.UI.WebControls
Imports System.ComponentModel

Public Class RainbowLabel
  Inherits System.Web.UI.WebControls.Label
```

```

Public Property EnableRainbowMode() As Boolean
    Get
        If ViewState("EnableRainbowMode") Is Nothing Then
            Return True
        Else
            Return Boolean.Parse(CStr(ViewState("EnableRainbowMode")))
        End If
    End Get
    Set(ByVal Value As Boolean)
        ViewState("EnableRainbowMode") = Value
    End Set
End Property

Protected Overrides Sub Render(ByVal output As HtmlTextWriter)
    If EnableRainbowMode Then
        output.Write(ColorizeString([Text]))
    Else
        output.Write([Text])
    End If
End Sub 'Render

Private Function ColorizeString(ByVal input As String) As String
    Dim output As New System.Text.StringBuilder(input.Length)
    Dim rand As Random = New Random(DateTime.Now.Millisecond)

    Dim i As Integer
    For i = 0 To input.Length - 1
        Dim red As Integer = rand.Next(0, 255)
        Dim green As Integer = rand.Next(0, 255)
        Dim blue As Integer = rand.Next(0, 255)

        output.Append("<font color=""#"")
        output.Append(Convert.ToString(red, 16))
        output.Append(Convert.ToString(green, 16))
        output.Append(Convert.ToString(blue, 16))
        output.Append(">"")
        output.Append(input.Substring(i, 1))
        output.Append("</font>"")
    Next i

    Return output.ToString()
End Function
End Class

```

To use this control, you need to do the following:

```

<%@ Page language="VB" %>
<%@ Register TagPrefix="AspNetCookbook" Namespace="AspNetCookbook"

```

```
Assembly="AspNetCookbook" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<html>
<head>
<title>03 Custom Controls - 02 Extending Existing Web Controls</title>
</head>
<body>

<form id="Form1" method="post" runat="server">
  <AspNetCookbook:RainbowLabel text="This is a rainbow colored test string"
    runat="server" /><br />
  <AspNetCookbook:RainbowLabel EnableRainbowMode="false"
    text="This is a test string" runat="server" />
</form>

</body>
</html>
```

Comments

This control creates a rainbow-ish pattern in the text it displays. This is accomplished by encapsulating each character in `` tags. The color on each character is randomized. Although perhaps not the most useful control, it does demonstrate how easy it is to extend the functionality of existing Web Controls. It is worth noting that the Label control should not be overlooked as a powerful control on which to build—the validation controls all inherit from the simple Label control.

See Also

Section 3.1, "Declaring a Simple Custom Control"

Developing ASP.NET Server Controls and Components by Nikhil Kothari and Vandana Datye (Microsoft Press; ISBN 0735615829)

3.3. Creating ViewState-Enabled Control Properties

You want your control's properties to retain their state using `ViewState`.

Technique

This example shows you how to create properties for your controls that retain their state using `ViewState`.

The `ViewStateControl` class:

```

Imports System
Imports System.Web.UI
Imports System.Web.UI.WebControls
Imports System.ComponentModel

Public Class ViewStateControl
    Inherits System.Web.UI.WebControls.WebControl

    Public Property [Text] () As String
        Get
            Dim _text As String = CStr(ViewState("Text"))
            If _text Is Nothing Then
                Return String.Empty
            Else
                Return _text
            End If
        End Get
        Set(ByVal Value As String)
            ViewState("Text") = Value
        End Set
    End Property

    Protected Overrides Sub Render(ByVal writer As HtmlTextWriter)
        writer.Write([Text])
    End Sub
End Class

```

To use this control, you need to do the following:

```

<%@ Page language="VB" %>
<%@ Register TagPrefix="AspNetCookbook" Namespace="AspNetCookbook"
Assembly="AspNetCookbook" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<html>
<head>
<title>Recipe0303</title>
</head>
<body>

<script language="VB" runat="server">
Sub Page_Load(Sender As Object, E As EventArgs)
    If Not IsPostBack Then
        Dim RandomGenerator As Random
        RandomGenerator = New Random(DateTime.Now.Millisecond)

        ViewStateControl1.Text = RandomGenerator.Next(1,100)
    End If

```

```
End Sub
</script>

<form id="Form1" method="post" runat="server">
  <AspNetCookbook:ViewStateControl id="ViewStateControl1" runat="server"/>
  <asp:linkbutton text="PostBack test" runat="server"/>
</form>

</body>
</html>
```

Comments

This control property called `Text` will retain its state on postbacks—as you see, it is quite easy to get properties to retain their state by using the `ViewState` property, and this technique is recommended for most Web Control properties.

See Also

Section 3.1, "Declaring a Simple Custom Control"

3. 4. Creating a Composite Control

You want to combine two or more Web Controls into a single composite custom control.

Technique

This example shows you how to create a very simple yet useful composite control. The control is a composition of a `TextBox` and a validator, and the control can be used to validate email addresses. The `EmailTextBox` class is as follows:

```
Imports System
Imports System.Web.UI
Imports System.Web.UI.WebControls
Imports System.ComponentModel
Imports System.ComponentModel.Design

Namespace AspNetCookbook
  Public Class EmailTextBox
    Inherits System.Web.UI.WebControls.WebControl
    Implements INamingContainer

    Private textBox As TextBox
    Private validator As RegularExpressionValidator
```

```

Public Property Text() As String
    Get
        EnsureChildControls()
        Return textBox.Text
    End Get
    Set(ByVal Value As String)
        EnsureChildControls()
        textBox.Text = Value
    End Set
End Property

```

```

Public Property ErrorMessage() As String
    Get
        EnsureChildControls()
        Return validator.ErrorMessage
    End Get
    Set(ByVal Value As String)
        EnsureChildControls()
        validator.ErrorMessage = Value
    End Set
End Property

```

```

Public Overrides ReadOnly Property Controls() As ControlCollection
    Get
        EnsureChildControls()
        Return MyBase.Controls
    End Get
End Property

```

```

Protected Overrides Sub CreateChildControls()
    Controls.Clear()

```

```

    textBox = New TextBox
    validator = New RegularExpressionValidator

```

```

    Controls.Add(textBox)
    Controls.Add(validator)

```

```

    textBox.ID = "Email1"
    validator.ControlToValidate = textBox.ID
    'A typical email address regular expression
    validator.ValidationExpression = "^[a-zA-Z0-9_\-\.]+\@(\[[0-9]{1,3}
\.[0-9]{1,3}\. [0-9]{1,3}\. | ([a-zA-Z0-9\-\ ]+\.)+))
([a-zA-Z]{2,4}|[0-9]{1,3}) (\[?)$"

```



```
    End Sub
  End Class
End Namespace
```

To use this control, you need to do the following:

```
<%@ Page language="VB" %>
<%@ Register TagPrefix="AspNetCookbook" Namespace="AspNetCookbook"
Assembly="AspNetCookbook" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<html>
<head>
<title>Recipe0304</title>
</head>
<body>

<form id="Form1" method="post" runat="server">
  <AspNetCookbook:EmailTextBox
    ID="EmailTextBox1"
    ErrorMessage="You must provide a valid email address!"
    runat="server"
  />
</form>

</body>
</html>
```

Comments

This control validates the input in the TextBox. If the input is not a valid email address an error message appears. Controls like this one are useful on pages that contain a lot of user input that needs to be validated. You can easily extend this control to use several validator types, allow the page developer to define his/her own validation expression, and so on.

See Also

Regular Expression Library—<http://regexlib.com/>

3.5. Creating a Data-bound Control

You want to create a custom control that supports data-binding.

Technique

This example shows you how to create a simple and original data-bound custom server control—a data-bound bulleted list. The `CustomBulletedList` class is as follows:

```
Imports System
Imports System.Web.UI
Imports System.Web.UI.WebControls
Imports System.ComponentModel
Imports System.Collections
Imports System.Text

Public Class CustomBulletedList
    Inherits System.Web.UI.WebControls.WebControl

    Private _html As New StringBuilder()
    Private _dataSource As IEnumerable

    Public Property DataSource() As IEnumerable
        Get
            Return _dataSource
        End Get
        Set(ByVal value As IEnumerable)
            _dataSource = value
        End Set
    End Property

    Private Sub CreateBulletedList()
        Dim dataSource As IEnumerable = Nothing

        Try
            dataSource = Me._dataSource
        Catch
        End Try

        If Not (dataSource Is Nothing) Then
            _html.Append("<ul>")
            Dim dataObject As Object
            For Each dataObject In dataSource
                _html.Append("<li>")
                _html.Append(dataObject)
                _html.Append("</li>")
            Next dataObject
            _html.Append("</ul>")
        End If
    End Sub
End Class
```

```

Public Overrides Sub DataBind()
    MyBase.OnDataBinding(EventArgs.Empty)

    CreateBulletedList()
End Sub

Protected Overrides Sub Render(ByVal output As HtmlTextWriter)
    output.Write(_html)
End Sub
End Class

```

To use this control, you need to do the following:

```

<%@ Page Language="VB" %>
<%@ Register TagPrefix="AspNetCookbook" Namespace="AspNetCookbook"
Assembly="AspNetCookbook" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<html>
<head>
<title>Data Bound Controls</title>
</head>
<body>
<script language="vb" runat="server">
Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
    listControl.DataSource = New String() {"Test 1", "Test 2", "Test 3"}
    listControl.DataBind()
End Sub
</script>

<AspNetCookbook:CustomBulletedList id="listControl" runat="server"/>

</body>
</html>

```

Comments

This control allows you to data-bind any data source that is derived from `IEnumerable` to it—the content will be listed in bulleted format. Overriding the `DataBind` method is the critical point in this control. When this method is called, it is trying to bind to the data source and call the `CreateBulletedList` method.

`CreateBulletedList` loops through all of the data objects in the data source and saves them to be outputted as bulleted lines.

By overriding the `Render` method, you can control the rendering of the control, and render it exactly as you see fit. You should always opt for this approach when performance is an issue (and when isn't it?), because it is much faster than overriding the `CreateChildControls` method.

One last point you should take note of is that you should always use a `StringBuilder` instead of a regular string as the HTML output source. Using regular string objects will *seriously* degrade the performance of your custom control.

See Also

Section 3.6, "Creating a Templated Control"

3.6. Creating a Templated Control

You want to create a control that supports the use of templates.

Technique

This example creates a simple templated control. It displays the current time on the server on which it runs and allows you to add dynamic text and so on.

First, the main control class, `DateTimeControl`:

```
Imports System
Imports System.Web.UI
Imports System.Web.UI.WebControls
Imports System.ComponentModel

Namespace AspNetCookbook
    <ToolboxData("<{0}:DateTimeControl runat=server></{0}:DateTimeControl>"), _
    ParseChildren(True) > _
    Public Class DateTimeControl
        Inherits Control
        Implements INamingContainer
        Private _template As ITemplate
        Private _container As DateTimeContainer
        Private _text As String

        <TemplateContainer(GetType(DateTimeContainer)) > _
        Public Overridable Property Template() As ITemplate
            Get
                Return _template
            End Get
            Set(ByVal Value As ITemplate)
                _template = Value
            End Set
        End Property

        Public Overridable ReadOnly Property Container() As DateTimeContainer
            Get
```

```

        Return _container
    End Get
End Property

Public Overridable Property Text() As String
    Get
        Return _text
    End Get
    Set(ByVal Value As String)
        _text = Value
    End Set
End Property

Public Overridable ReadOnly Property DateTime() As String
    Get
        Return System.DateTime.Now.ToShortTimeString()
    End Get
End Property

Protected Overrides Sub OnDataBinding(ByVal e As EventArgs)
    EnsureChildControls()
    MyBase.OnDataBinding(e)
End Sub

Protected Overrides Sub CreateChildControls()
    If Not (Template Is Nothing) Then
        _container = New DateTimeContainer(Text, DateTime)
        Template.InstantiateIn(Container)
        Controls.Add(Container)
    Else
        Controls.Add(New LiteralControl("<# Container.Text %>"))
    End If
End Sub
End Class

```

Now let's take a look at the container control `DateTimeContainer`. It enables you to use `<# Container.Text %>` and so forth.

```

Public Class DateTimeContainer
    Inherits Control
    Implements INamingContainer
    Private _text As String
    Private _dateTime As String

    Public Sub New(ByVal text As String, ByVal dateTime As String)

```

```

        Me._text = text
        Me._dateTime = dateTime
    End Sub 'New

    Public ReadOnly Property Text() As String
        Get
            Return _text
        End Get
    End Property

    Public ReadOnly Property DateTime() As String
        Get
            Return _dateTime
        End Get
    End Property
End Class
End Namespace

```

Comments

This control allows you to add a template and thus choose how you want the data to be presented. The control is built using two classes—`DateTimeControl` is the actual control you add to the page and `DateTimeContainer` is the container control that holds the actual template data. Use of a container class is, strictly speaking, not necessary if you don't want to use custom properties. If you just want to display static content, you can instantiate the control in a `Panel` control or a similar control.

See Also

Section 3.4, "Creating a Composite Control"

Section 3.5, "Creating a Data-bound Control"

3.7. Dynamically Adding Controls to a Web Form

You want to programmatically insert controls into a Web Form.

Technique

You can add controls to any class that exposes a `Controls()` collection property by using the `Class.Controls.Add()` method. In this example, the user chooses which control should be added to the page. The `Placeholder` control can be used to specify

exactly where on a Web Form you want your new control to appear, but otherwise is not necessary for this technique to work.

The ASPX page is as follows:

```
<form id="Recipe0307" method="post" runat="server">
  <asp:DropDownList id="DropDownList1" runat="server">
    <asp:ListItem Value="Simple Text Control (Recipe0301)">
      Simple Text Control (Recipe0301)</asp:ListItem>
    <asp:ListItem Value="RainbowLabel (Recipe0302)">RainbowLabel
      (Recipe0302)</asp:ListItem>
  </asp:DropDownList>
  <asp:Button id="Button1" runat="server" Text="Select"
    onclick="Button1_Click"></asp:Button>
  <p>
    <asp:Placeholder id="Placeholder1" runat="server"></asp:Placeholder>
  </p>
</form>
```

In `<script runat="server" />` block or codebehind:

```
Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
  \ Initialize Placeholder
  Placeholder1.Controls.Clear()

  Select Case DropDownList1.SelectedIndex
    Case 0:
      Dim SimpleText As ASPNetCookbook.Recipe0301vb = _
        New ASPNetCookbook.Recipe0301vb()
      SimpleText.Text = "This is the simple text control from Recipe 3.01."
      Placeholder1.Controls.Add(SimpleText)
      Exit Select
    Case 1:
      Dim RainbowText As ASPNetCookbook.RainbowLabel = _
        New ASPNetCookbook.RainbowLabel()
      RainbowText.Text = "This is the RainbowText control from Recipe 3.02."
      Placeholder1.Controls.Add(RainbowText)
      Exit Select
  End Select
End Sub
```

Comments

Dynamically creating and inserting controls onto a page is a very powerful technique that you can easily extend to create flexible page architectures.

3.8. Using the Treeview IE Web Control

You want to implement a tree view using the TreeView IE Web Control.

Technique

The first step is to install the IE Web Controls from Microsoft. The link to download and install these controls is as follows: http://msdn.microsoft.com/downloads/samples/internet/default.asp?url=/downloads/samples/internet/asp_dot_net_servercontrols/webcontrols/default.asp

The next step is to add a reference to the controls on your page:

```
<%@ Register TagPrefix="mytree" Namespace="Microsoft.Web.UI.WebControls"
Assembly="Microsoft.Web.UI.WebControls" %>
<%@ import namespace="Microsoft.Web.UI.WebControls" %>
```

Then you add the TreeView control to your page:

```
<mytree:treeview runat="server" id="myTree">
</mytree:treeview>
```

The last step is to add the node information to the TreeView. This is the list of items that will appear when you expand the tree:

```
<mytree:treeview runat="server" id="myTree">
  <mytree:treenode text="Cars">
    <mytree:treenode text="Ford" />
    <mytree:treenode text="Toyota" />
    <mytree:treenode text="Infiniti" />
  </mytree:treenode>
</mytree:treeview>
```

Here, we created one node called `Cars` and then added nodes within that node for different car manufacturers.

Comments

The TreeView IE Web Control is a very powerful and easy-to-use control. In this example, you saw how easy it was to create an explorer-like expandable tree. When adding nodes to the TreeView, you use a very XML-like approach whereby the nodes that are under a parent node are contained inside the parent node tag. If you encounter errors when trying to add a reference to your page, or you receive errors indicating that the Web Controls cannot be found, copy the Web Controls DLL to the `bin` directory of your application.

See Also

Basics of the TreeView Control—

<http://www.aspalliance.com/jamesavery/webcontrols/treeviewp1.aspx>

Section 3.11, "Data-binding a TreeView Control"

Overview of Treeview Control—

<http://msdn.microsoft.com/workshop/webcontrols/overview/treeview.asp>

3.9. Using the TabControl and PageView IE Web Controls

You want to implement a tabbed interface or a wizard-like form using the IE Web Controls TabControl and PageView.

Technique

The first step is to install the IE Web Controls from Microsoft. The link to download and install these controls is as follows: http://msdn.microsoft.com/downloads/samples/internet/default.asp?url=/downloads/samples/internet/asp_dot_net_servercontrols/webcontrols/default.asp

Next, you need to add a reference to the Web Controls on your page:

```
<%@ Register TagPrefix="ieControls"
Namespace="Microsoft.Web.UI.WebControls"
Assembly="Microsoft.Web.UI.WebControls" %>
<%@ import namespace="Microsoft.Web.UI.WebControls" %>
```

Then you add your tabstrip. This is the control that generates the tabs that users click to change pages:

```
<IECONTROLS:TABSTRIP id="myTabStrip" runat="server" TargetID="myMultiPage"
Orientation="Vertical">
    <IECONTROLS:Tab Text="Tab 1"></IECONTROLS:Tab>
    <IECONTROLS:TabSeparator></IECONTROLS:TabSeparator>
    <IECONTROLS:Tab Text="Tab 2"></IECONTROLS:Tab>
    <IECONTROLS:TabSeparator DefaultStyle="height:100%;">
    </IECONTROLS:TabSeparator>
</IECONTROLS:TABSTRIP>
```

Lastly, you need to add your multipage control that contains the different pages that will be viewed when users select each tab:

```
<IECONTROLS:MULTIPAGE id="myMultiPage" runat="server">
    <IECONTROLS:PAGEVIEW>
        Page 1
```

```

</IECONTROLS:PAGEVIEW>
<IECONTROLS:PAGEVIEW>
    Page 2
</IECONTROLS:PAGEVIEW>
</IECONTROLS:MULTIPAGE>

```

Comments

Notice that there is a property on the tabstrip control called `TargetID`. This must be set to the ID of the multipage control that will be working with the tabstrip. When the users click on the tab, the tabstrip will then tell the `PageView` to show the corresponding page. When working with the multipage control, you can add any type of ASP.NET or HTML between the opening and closing `PageView` tags. In this example, the tabstrip is running vertically, but you can also set this value to horizontal. To do so, you simply need to set the `Orientation` property to `horizontal`.

See Also

Overview of `TabStrip` Control—

<http://msdn.microsoft.com/workshop/webcontrols/overview/tabstrip.asp>

Overview of `Multipage` Control—

<http://msdn.microsoft.com/workshop/webcontrols/overview/multipage.asp>

3.10. Using the `ToolBar` IE Web Control

You want to implement a toolbar using the `ToolBar` IE Web Control.

Technique

The first step is to install the IE Web Controls from Microsoft. The link to download and install these controls is as follows: http://msdn.microsoft.com/downloads/samples/internet/default.asp?url=/downloads/samples/internet/asp_dot_net_servercontrols/webcontrols/default.asp

Next, you need to add a reference to the Web Controls on your page:

```

<%@ Register TagPrefix="ieControls"
Namespace="Microsoft.Web.UI.WebControls"
Assembly="Microsoft.Web.UI.WebControls" %>
<%@ import namespace="Microsoft.Web.UI.WebControls" %>

```

Then, you add the toolbar to your page:

```

<iecontrols:ToolBar id="myToolBar" runat="server">
</ie:ToolBar>

```

Lastly, you add the buttons that you want to appear on your toolbar:

```
<iecontrols:ToolBar id="myToolBar" runat="server">
  <iecontrols:ToolBarButton Text="First Button" />
  <iecontrols:ToolBarSeparator />
  <iecontrols:ToolBarButton Text="<b>Second Button</b>" />
</iecontrols:ToolBar>
```

Comments

The toolbar control is a quick way to get a nice-looking toolbar working on your site. Notice in this example that you can even add rich content to the button text. This enables you to use HTML on your buttons. You can also set the `ImageUrl` property of the button tag to use an image as your button.

See Also

Overview of ToolBar Control—

<http://msdn.microsoft.com/workshop/webcontrols/overview/toolbar.asp>

3.11. Data-binding a TreeView Control

You want to populate a treeview with data by using XML and the TreeView IE Web Control.

Technique

If you have not worked with the TreeView control before, refer to section 3.8 for basic TreeView information. First, you need to add the TreeView to your page and set the `treenodesrc` property to the XML file that you'll be creating.

```
<mytree:treeview runat="server" id="myTree" treenodesrc="cars.XML">
</mytree:treeview>
```

The next thing you need to do is create your XML file that you will use to bind the TreeView to:

```
<?xml version="1.0" encoding="utf-8" ?>
<TREENODES>
  <treenode text="Cars">
    <treenode text="Ford" />
    <treenode text="Toyota" />
    <treenode text="Infiniti" />
  </treenode>
</TREENODES>
```

When the page loads, the TreeView will generate its nodes from the XML file.

Comments

It is not possible to directly data-bind a dataset to the TreeView Web Control. The data must first be translated to XML. You can use `response.write`, which writes the XML, or use XSLT to transform the dataset into the correct XML. When using XSLT, you can specify the XSLT file using the `TreeNodeXsltSrc` property of the TreeView. In addition to using XML to define the actual nodes of the TreeView, you can also use XML to define the style types by using the `TreeNodeTypeSrc` property of the TreeView control.

See Also

Overview of Treeview Control—

<http://msdn.microsoft.com/workshop/webcontrols/overview/treeview.asp>

Creating Dynamic TreeViews—

<http://www.aspalliance.com/jamesavery/webcontrols/treeviewp2.aspx>

3.12. Installing a Component in the Global Assembly Cache (GAC)

You want to access your custom control from any application on a server by registering its assembly in the Global Assembly Cache, or GAC.

Technique

The Global Assembly Cache in the .NET Framework provides a central place for registering assemblies. Registered assemblies are then available to all applications, including the development environments like Visual Studio.NET and Web Matrix. The process of adding an assembly to the Global Assembly Cache can be compared to the process of registering COM components in the server, as is done in the case of Windows DNA applications.

The first step involved in adding an assembly is to *strong-name* the assembly. A strong name is basically assigned to an assembly or a component to distinguish it from other assemblies and components existing in the GAC. A strong name consists of an assembly identity (name, version, and so on), a public key, and a digital signature.

Assigning a strong name to an assembly ensures that it is unique, has version protection, and has code integrity. Assigning a strong name to an assembly is not a difficult task. You can use the `sn.exe` utility to generate strong names, which are then added to the code of the assembly. For example, to create a strong name for an assembly named `sample.dll`, you would write the following in the command prompt:

```
c:\MyAssembly>sn -k sample.snk
```

This would generate a strong name keypair and store it in a file named `sample.snk`. The file extension can be anything, but `.SNK` is normally used as a convention. The `-k` option creates a strong name keypair. There are other options that you can search for in MSDN.

The second step is to associate the generated strong name file with the assembly. Add the following code to your assembly to associate the strong name. By default, Visual Studio .NET projects include skeleton declarations of these attributes in the `AssemblyInfo` file.

```
Imports System.Reflection

<assembly: AssemblyKeyFile("sample.snk")>
```

Note here that the information regarding the file containing the strong name keypair is placed in the code file before any namespace declaration. Also, you must import the `System.Reflection` namespace in order for the statement to work, otherwise the compiler will not recognize the `<assembly: AssemblyKeyFile("sample.snk")>` statement.

After compiling the assembly with the statements containing the strong name information being added to it, you now have to place the assembly into the GAC. You can do this manually by simply copying and pasting the assembly into the GAC, which is located at `C:\WINNT\ASSEMBLY\`.

You can also use the `gacutil.exe` utility, installed with the .NET Framework. In order to add an assembly, you write the following on the command prompt:

```
gacutil /i sample.dll
```

The `/i` option is for installation. In order to remove an assembly from the GAC, you can use the same utility, as follows:

```
gacutil /u sample.dll
```

The `/u` option is for uninstalling or removing an assembly from the cache. Typing `gacutil /?` lists all of its options.

See Also

Section 3.1, "Declaring a Simple Custom Control"

Section 3.2, "Creating a Composite Control"

Command-Line Compilation—<http://aspalliance.com/hrmalik/articles/2002/200211/20021101.aspx>, Haroon Rasheed Malik

The .NET Assemblies—<http://aspalliance.com/hrmalik/articles/2002/200202/20020201.aspx>, Haroon Rasheed Malik

