

Chapter

16

Developing the ASP.NET TabView Control

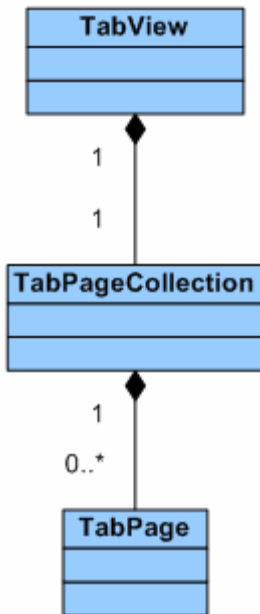
Overview

Another useful control that was not available as a component in the ASP.NET control framework is the tabbed control. A tabbed control displays multiple tabs, of which each can be clicked to bring focus to a new tab page. The tabs are analogous to dividers in a notebook, where flipping a divider will yield the divider page.

The goal in this chapter is to implement an ASP.NET server control named TabView that simulates a tabbed control. The control will support both vertical tabs and horizontal tabs. Vertical tabs are those that can be found on the Visual Studio .NET 2003 Toolbox. In this chapter, I will demonstrate rendering, state management, parsing, and design-time support.

Step 1: The Architecture

The model for the TabView control is very simple. It can be illustrated as follows:



Simply stated, the *TabView* class contains a *TabCollection* instance. Each item in the collection is of type *TabPage*.

Step 2: The User Interface

Now that you understand the model, you can create the user interface. When developing server controls, you will sometimes need to utilize the HTML designer and the “View In Browser” function for brainstorming. In the case of this example, the best way to design your control is to first design it in an HTML designer separate from your project. So let’s do that by creating an HTML file in Visual Studio .NET. 2003

First start with horizontal rendering. Render the horizontal view as an HTML table with two rows and one column. The first row will be used to display the tab page headers while the second row will be used to display the tab page views, as shown here:



Now render the tab page headers by creating five columns and setting the tab page view to span five columns. Here is the output:



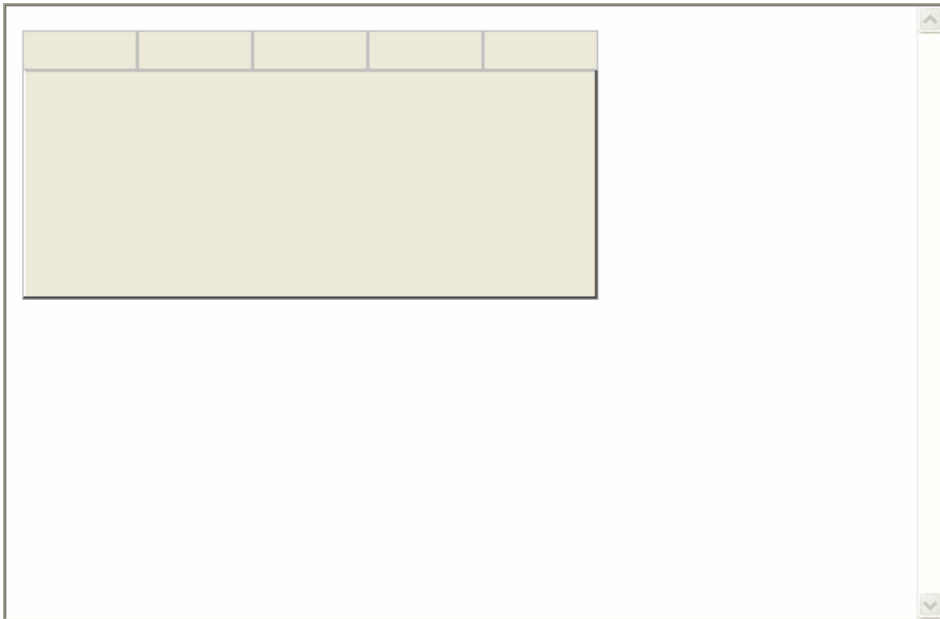
Before going further, glance at the following HTML code used to generate the output:

```
<TABLE id="Table1" height="168" cellSpacing="1" cellPadding="1"
width="360" border="1">
  <TR>
    <TD height="25"></TD>
    <TD height="25"></TD>
    <TD height="25"></TD>
    <TD height="25"></TD>
    <TD height="25"></TD>
  </TR>
  <TR>
    <TD colspan="5"></TD>
  </TR>
</TABLE>
```

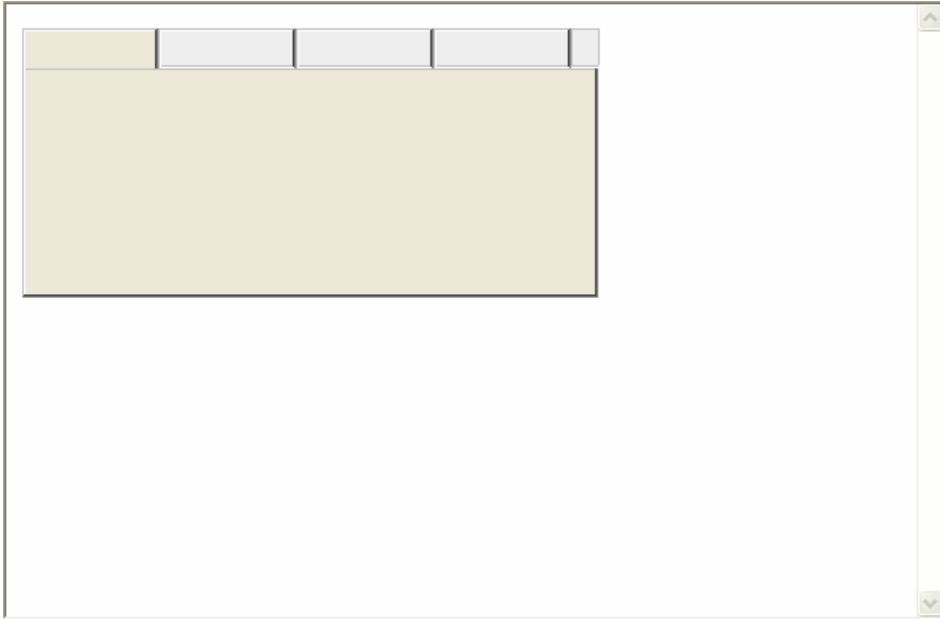
You need to now make the previous output look more like a true tabbed control. Start by changing the background color, border width, cell spacing, and cell padding of the table, and by modifying the borders of the tab page view, as shown here:

```
<TABLE id="Table1" height="168" cellSpacing="0" cellPadding="0"
width="360" border="0" bgcolor="menu">
  <TR>
    <TD height="25"></TD>
    <TD height="25"></TD>
    <TD height="25"></TD>
    <TD height="25"></TD>
    <TD height="25"></TD>
  </TR>
  <TR>
    <TD colspan="5" style="BORDER-RIGHT: white thin outset;
BORDER-LEFT: white thin outset; BORDER-TOP-STYLE: none; BORDER-BOTTOM:
white thin outset"></TD>
  </TR>
</TABLE>
```

The previous code will produce the following output:



The tab page view now has more of a three-dimensional look. Similarly, you can do the same to the tab page headers. Also, differentiate the selected tab page header from the others through border modifications and colors, as shown here:

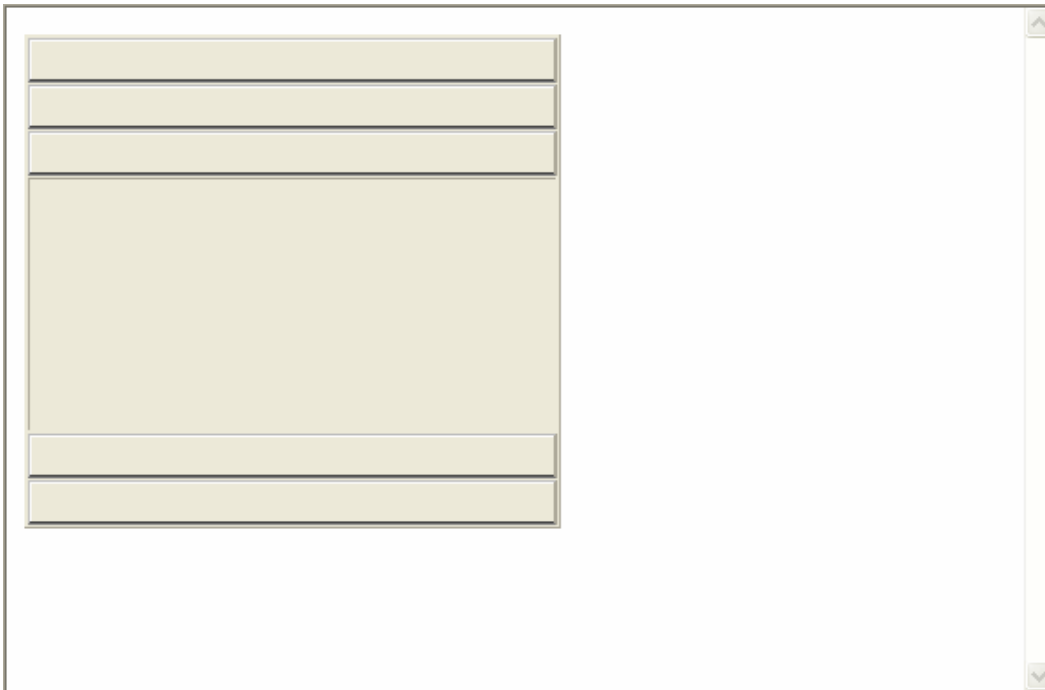


You now have some handy HTML code to help in rendering the TabView control horizontally. So let's do the same for vertical rendering. To jump right into things, here is the code for the vertical tab with the output following:

```
<TABLE id="Table1" style="WIDTH: 300px; HEIGHT: 237px" cellSpacing="1"
cellPadding="1"
width="300" bgColor="menu" border="1">
  <TR>
    <TD style="BORDER-RIGHT: thin outset; BORDER-TOP: white thin
outset; BORDER-LEFT: white thin outset; BORDER-BOTTOM: white thin outset;
HEIGHT: 15px"></TD>
  </TR>
  <TR>
    <TD style="BORDER-RIGHT: thin outset; BORDER-TOP: white thin
outset; BORDER-LEFT: white thin outset; BORDER-BOTTOM: white thin outset;
HEIGHT: 15px"></TD>
  </TR>
  <TR>
    <TD style="BORDER-RIGHT: thin outset; BORDER-TOP: white thin
outset; BORDER-LEFT: white thin outset; BORDER-BOTTOM: white thin outset;
HEIGHT: 15px"></TD>
  </TR>
  <TR>
    <TD style="HEIGHT: 142px"></TD>
  </TR>
```

```
<TR>
  <TD style="BORDER-RIGHT: thin outset; BORDER-TOP: white thin
outset; BORDER-LEFT: white thin outset; BORDER-BOTTOM: white thin outset;
HEIGHT: 15px"></TD>
</TR>
<TR>
  <TD style="BORDER-RIGHT: thin outset; BORDER-TOP: white thin
outset; BORDER-LEFT: white thin outset; BORDER-BOTTOM: white thin outset;
HEIGHT: 15px"></TD>
</TR>
</TABLE>
```

Here is the output:

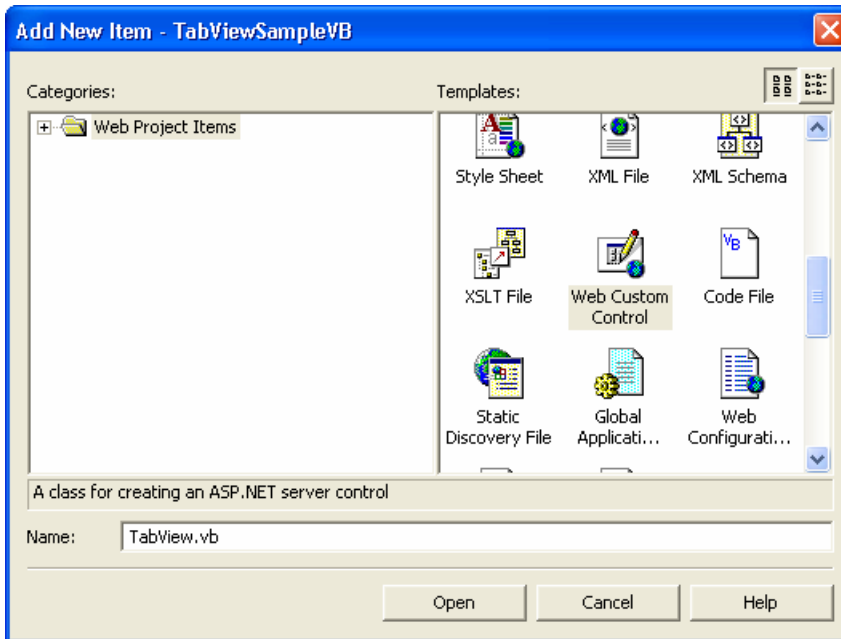


The vertical view of the TabView control uses the same three-dimensional effects that are used by the horizontal one. Now that you have everything you need from a user interface point of view, you can begin the implementation.

Step 3: Runtime Implementation

For the implementation phase, you should concentrate on four specific areas: implementing the model, providing parsing and persistence support, providing the rendering logic, and maintaining the state of the control between postbacks. The first thing you need to do is implement the model. You know that the TabView control should contain a collection of tab pages and that it can be rendered both vertically and horizontally. Therefore, create the following three classes: *TabView*, *TabPageCollection*, and *TabPage*. Also create one enumeration type, *TabViewOrientation*, which will indicate how the control is to be rendered. These classes and enumeration type simply reflect the model.

Go ahead and create a new ASP.NET Web Application project named TabViewSampleVB using Visual Studio .NET 2003. Add a new Web Custom Control named TabView to the project, as shown here:



Now change the *TabView* class so that it derives from *Control* instead of *WebControl*. Remove the *Text* property and its member variable, as well as the *DefaultPropertyAttribute* object that specifies the *Text* property as the default property. Also remove the implementation of the *Render* method, since you will add custom rendering logic later. Here is the class definition after the modifications:

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

<ToolboxData("<{0}:TabView runat=server></{0}:TabView>")> Public Class
TabView
    Inherits Control

    Protected Overrides Sub Render(ByVal output As
System.Web.UI.HtmlTextWriter)
    End Sub
End Class
```

Now define the *TabViewOrientation* enumeration type as a nested type within the *TabView* class. Also add a property named *Orientation* to the class. Here is the code:

```
Public Enum TabViewOrientation
    Horizontal
    Vertical
End Enum

Public Property Orientation() As TabViewOrientation
    Get
        If (TypeOf ViewState("Orientation") Is TabViewOrientation) Then
            Return ViewState("Orientation")
        Else
            Return TabViewOrientation.Horizontal
        End If
    End Get
    Set(ByVal Value As TabViewOrientation)
        ViewState("Orientation") = Value
    End Set
End Property
```

Go ahead and add *Width* and *Height* properties so that the *TabView* control can be sized, as well as a *SelectedIndex* property that specifies the numeric index of the selected tab. Here is the code:

```
Public Property Width() As Unit
    Get
        If (TypeOf ViewState("Width") Is Unit) Then
            Return ViewState("Width")
        Else
            Return Unit.Percentage(100)
        End If
    End Get
    Set(ByVal Value As Unit)
        ViewState("Width") = Value
    End Set
End Property

Public Property Height() As Unit
    Get
        If (TypeOf ViewState("Height") Is Unit) Then
            Return ViewState("Height")
        Else
            Return Unit.Percentage(100)
        End If
    End Get
    Set(ByVal Value As Unit)
        ViewState("Height") = Value
    End Set
End Property

<Browsable(False)> _
Public Property SelectedIndex() As Integer
    Get
        If (TypeOf ViewState("SelectedIndex") Is Integer) Then
            Return ViewState("SelectedIndex")
        Else
            Return 0
        End If
    End Get
    Set(ByVal Value As Integer)
        ViewState("SelectedIndex") = Value
    End Set
End Property
```

Notice that the previous properties use the *ViewState* object to store their values. This allows the control's properties to be remembered across postbacks.

Now you can define the *TabPage* class, which should also derive from *Control*. You only need one member of this class, which is a property named *HeaderText* of type *String*. This property

represents the text that will be displayed on a tab page header. Here is the definition of this class:

```
Public Class TabPage
    Inherits Control

    Public Property HeaderText() As String
        Get
            If (TypeOf ViewState("HeaderText") Is String) Then
                Return ViewState("HeaderText")
            Else
                Return "TabPageHeader"
            End If
        End Get
        Set(ByVal Value As String)
            ViewState("HeaderText") = Value
        End Set
    End Property
End Class
```

Next, create a class named *TabPageCollection* that will be used to store *TabPage* instances. Here is the class:

```
Public Class TabPageCollection
    Inherits CollectionBase

    Default Public ReadOnly Property Item(ByVal pageIndex As Integer) As
TabPage
        Get
            Return MyBase.List(pageIndex)
        End Get
    End Property

    Public Sub Add(ByVal page As TabPage)
        MyBase.List.Add(page)
    End Sub

    Public Function IndexOf(ByVal page As TabPage) As Integer
        Return MyBase.List.IndexOf(page)
    End Function
End Class
```

You also need to define a property and a corresponding member variable in the *TabView* class so that it is aware of its collection of *TabPage* instances. Add the following code to the *TabView* class:

```
Private _tabPages As TabPageCollection = New TabPageCollection
...

Public ReadOnly Property TabPages() As TabPageCollection
    Get
        Return _tabPages
    End Get
End Property
```

Now, you need to support parsing and persistence so that tab pages added through the designer can be saved to and retrieved from an ASP.NET page correctly. This can be accomplished using the *PersistChildrenAttribute*, *PersistenceModeAttribute*, and the *ParseChildrenAttribute* classes. They should be applied as follows:

```
<ToolboxData("<0:TabView runat=server></0:TabView>"), _
PersistChildren(False), _
ParseChildren(True, "TabPages")> _
Public Class TabView
    Inherits Control
    ...

    <PersistenceMode(PersistenceMode.InnerDefaultProperty)> _
    Public ReadOnly Property TabPages() As TabPageCollection
    Get
        Return _tabPages
    End Get
End Property

...
End Class
```

The *ParseChildrenAttribute* object tells the control parser to construct the *TabPages* property using any nested elements of the *TabView* control's element on the page. Because of this attribute, you do not need to implement and specify a custom control builder. The

PersistChildrenAttribute object tells the control persister not to persist the child controls of the TabView control, but instead persist the properties of the TabView control. The *PersistenceModeAttribute* object tells the control persister to write the items in the *TabPage* collection as nested elements. Note that the *InnerDefaultProperty* member is used instead of the *InnerProperty* member because the *DefaultProperty* overload of the *ParseChildrenAttribute* class is used. Both attributes must be able to work together for consistent parsing and persistence.

Now you need to override the *CreateChildControls* method so that the TabView control builds its hierarchy using only TabPage controls. Here is the overridden method:

```
Protected Overrides Sub CreateChildControls()  
    Me.Controls.Clear()  
    Dim tabPage As TabPage  
    For Each tabPage In Me.TabPages  
        Me.Controls.Add(tabPage)  
    Next  
End Sub
```

Because this is a composite control, you need to add the *INamingContainer* interface to the inheritance list of the *TabView* class so that the names of the child controls will be unique. You also need to do the same for the *TabPage* class. Here are the new class declarations:

```
Public Class TabView  
    Inherits Control  
    Implements INamingContainer
```

```
Public Class TabPage  
    Inherits Control  
    Implements INamingContainer
```

Because you know the styles that will be involved in rendering the control, go ahead and create a style sheet file named *TabView.css* for storing these styles and set its Build Action property

to Embedded Resource. This allows the resource to be extracted at run time. Here is what I came up with:

```
.TabViewHorizontal_Header
{
    border-left:thin outset white;
    border-right:thin outset white;
    border-top:thin outset white;
    border-bottom:thin inset window;
    padding:0,0,0,5;
    margin:0px;
    height:20px;
    width:25%;
    font-size:smaller;
    font-weight:400;
    font-family:Menu, Verdana, Arial;
    color:menutext;
    background-color:eeeeee;
}

.TabViewHorizontal_HeaderSelected
{
    border-left:thin outset white;
    border-right:thin outset white;
    border-top:thin outset white;
    padding:0,0,0,5;
    margin:0px;
    height:20px;
    width:25%;
    font-size:smaller;
    font-weight:bold;
    font-family:Menu, Verdana, Arial;
    color:menutext;
    background-color:menu;
}

.TabViewHorizontal_Content
{
    border-top:medium-none;
    border-bottom:thin outset white;
    border-left:thin outset white;
    border-right:thin outset white;
    padding:0px;
    margin:0px;
    height:100%;
    width:100%;
    background-color:menu;
    cursor:auto;
}

.TabViewHorizontal_Space
```

```
{
    border-bottom:thin inset window;
    font:8pt Verdana;
    width:100%;
    padding:0px;
    background-color:menu;
}

.TabViewVertical_Header
{
    border-top:thin outset white;
    border-bottom:thin outset;
    border-left:thin outset white;
    border-right:thin outset;
    padding:0,0,0,5;
    margin:0px;
    height:20px;
    font-size:smaller;
    font-weight:400;
    font-family:Menu, Verdana, Arial;
    color:menutext;
    background-color:menu;
}

.TabViewVertical_HeaderSelected
{
    border-top:thin outset white;
    border-bottom:thin outset;
    border-left:thin outset white;
    border-right:thin outset;
    padding:0,0,0,5;
    margin:0px;
    height:20px;
    font-size:smaller;
    font-weight:bold;
    font-family:Menu, Verdana, Arial;
    color:menutext;
    background-color:menu;
}
```

Add the following method to the *TabView* class so that it can extract the style sheet file at run time:

```
Private Sub ExtractServerResources(ByVal resourceNameSpace As String,
ByVal ParamArray resources As String())
    Dim resource As String
    For Each resource In resources
```

```

        Dim fullExtractedFileName As String =
Page.Server.MapPath(resource)
        Dim cacheKey As String = Me.GetType().Name + "_" +
fullExtractedFileName
        Dim fullResourceName As String = String.Format("{0}.{1}",
resourceNamespace, resource)

        ' Check to see if the resource has already been extracted.
        ' If it has, make sure it is the most recent one by checking the
assembly's date.
        ' If the assembly's date has changed, it's possible that the
resource has changed.

        Dim alreadyExtracted As Boolean =
File.Exists(fullExtractedFileName)
        Dim thisAssembly As [Assembly] =
[Assembly].GetExecutingAssembly()
        Dim assemblyPath As String =
thisAssembly.CodeBase.Substring("file:///".Length)
        Dim assemblyDateTime As DateTime =
File.GetLastWriteTime(assemblyPath)
        Dim cacheValue As Object = HttpRuntime.Cache(cacheKey)
        If (alreadyExtracted AndAlso Not cacheValue Is Nothing) Then
            Dim cachedDateTime As DateTime = cacheValue
            If (DateTime.Compare(cachedDateTime, assemblyDateTime) >= 0)
Then
                GoTo Continue
            End If
        End If

        Dim inputFile As Stream =
thisAssembly.GetManifestResourceStream(fullResourceName)
        System.Diagnostics.Debug.Assert(Not inputFile Is Nothing,
"inputFile is null", "The resource file is not embedded in the assembly
at the specified path.")
        Dim outputFile As Stream = New FileStream(fullExtractedFileName,
FileStream.Create, FileAccess.Write)

        Dim reader As StreamReader = New StreamReader(inputFile)
        Dim writer As StreamWriter = New StreamWriter(outputFile)

        writer.Write(reader.ReadToEnd())

        reader.Close()
        writer.Close()

        outputFile.Close()
        inputFile.Close()

        assemblyPath = assemblyPath.Replace("/", "\")
        Dim dependencies As String() = {assemblyPath}

```

```

        HttpRuntime.Cache.Insert(cacheKey, assemblyDateTime, New
CacheDependency(dependencies))
Continue:
    Next
End Sub

```

You also need to add *Imports* statements for the following namespaces at the beginning of the source file: *System.Reflection*, *System.IO*, *System.Web*, and *System.Web.Caching*. Because this method saves data to the Web server, the account used to run your Web application will need Write permissions on the directory that will contain the extracted resource. The extraction process writes to disk only when the assembly's date changes. An assembly date change reflects a possible resource change. If you do not wish or know how to enable Write access, you can always extract the resources manually using some of the .NET Framework tools and then place the extracted file in the directory of the Web application that will host the control. Note that you could have easily hard-coded all the styles in the *Render* method, but adding them to the resources allows for easy modification.

Now create a private method named *RenderSupportScripts* for rendering scripts and style sheets, as shown here:

```

Private Sub RenderSupportScripts (ByVal output As HtmlTextWriter)
    ExtractServerResources (Me.GetType ().Namespace, "TabView.css")

    ' Apply the CSS link.
    output.WriteLine("<LINK REL=""stylesheet"" TYPE=""text/css""
    HREF=""TabView.css"">")
End Sub

```

Now implement the *Render* method of the control to immediately call the *RenderSupportScripts* method, as shown here:

```

Protected Overrides Sub Render (ByVal output As
System.Web.UI.HtmlTextWriter)
    If (Not Page Is Nothing AndAlso Not
Page.IsClientScriptBlockRegistered (Me.GetType ().FullName)) Then

```

```

        Page.RegisterClientScriptBlock(Me.GetType().FullName, "")
        RenderSupportScripts(output)
    End If
End Sub

```

This code first checks to see if the *Link* element that specifies the style sheet file has already been rendered by another control or another instance of this control. If it has not, it extracts the style sheet file and renders the *Link* element. This functionality is enabled by the call to the *IsClientScriptBlockRegistered* method of the *Page* class. Now add two more private methods: *RenderControlHorizontal* and *RenderControlVertical*. Call these methods from the *Render* method, as shown here:

```

    Protected Overrides Sub Render(ByVal output As
System.Web.UI.HtmlTextWriter)
        If (Not Page.IsNothing AndAlso Not
Page.IsClientScriptBlockRegistered(Me.GetType().FullName)) Then
            Page.RegisterClientScriptBlock(Me.GetType().FullName, "")
            RenderSupportScripts(output)
        End If

```

```

    If (TabPage.Count > 0) Then
        SelectedIndex = 0

        If (Orientation = TabViewOrientation.Vertical) Then
            RenderControlVertical(output)
        ElseIf (Orientation = TabViewOrientation.Horizontal) Then
            RenderControlHorizontal(output)
        End If
    End If
End Sub

```

```

Private Sub RenderControlVertical(ByVal output As HtmlTextWriter)
    ' TabView
    Dim table As HtmlTable = New HtmlTable
    table.ID = Me.UniqueID
    table.Width = Me.Width.ToString()
    table.Height = Me.Height.ToString()
    table.CellSpacing = 1
    table.CellPadding = 0
    table.Border = 1
    table.BgColor = "Menu"
    table.Style.Add("cursor", "default")

    ' Render each tab page in the TabPageCollection.

```

```

Dim tabPage As TabPage
For Each tabPage In Me.TabPages
    Dim displayStyle As String = "none"
    If (Me.SelectedIndex = Me.TabPages.IndexOf(tabPage)) Then
        displayStyle = "block"
    End If

    ' tabPage Header
    Dim tr As HtmlTableRow = New HtmlTableRow
    Dim td As HtmlTableCell = New HtmlTableCell
    If (Me.SelectedIndex = Me.TabPages.IndexOf(tabPage)) Then
        td.Attributes.Add("class", "TabViewVertical_HeaderSelected")
    Else
        td.Attributes.Add("class", "TabViewVertical_Header")
    End If
    td.Attributes.Add("orientation",
Me.Orientation.ToString().ToLower())
    td.Controls.Add(New LiteralControl(tabPage.HeaderText))
    tr.Controls.Add(td)
    table.Controls.Add(tr)

    ' tabPage View
    If (Me.SelectedIndex = Me.TabPages.IndexOf(tabPage)) Then
        tr = New HtmlTableRow
        td = New HtmlTableCell
        td.Attributes.Add("class", "TabViewVertical_Content")
        td.Controls.Add(tabPage)
        tr.Controls.Add(td)
        tr.Style.Add("display", displayStyle)
        table.Controls.Add(tr)
    End If
Next

table.RenderControl(output)
End Sub

Private Sub RenderControlHorizontal(ByVal output As HtmlTextWriter)
    ' TabView
    Dim table As HtmlTable = New HtmlTable
    Table.ID = Me.UniqueID
    Table.Width = Me.Width.ToString()
    Table.Height = Me.Height.ToString()
    Table.CellSpacing = 0
    Table.CellPadding = 0
    Table.Border = 0
    Table.BgColor = "Menu"
    Table.Style.Add("cursor", "default")

    Dim tr As HtmlTableRow = New HtmlTableRow

    Dim widthPercent As Integer = 15
    If (Me.TabPages.Count * widthPercent > 100) Then

```

```

        widthPercent = 100 / Me.TabPages.Count
    End If

    ' Render each tab page in the TabPageCollection.
    Dim tabPage As TabPage
    For Each tabPage In Me.TabPages
        ' tabPage Header
        Dim td As HtmlTableCell = New HtmlTableCell
        If (Me.SelectedIndex = Me.TabPages.IndexOf(tabPage)) Then
            td.Attributes.Add("class", "TabViewHorizontal_HeaderSelected")
        Else
            td.Attributes.Add("class", "TabViewHorizontal_Header")
        End If
        td.Attributes.Add("orientation",
Me.Orientation.ToString().ToLower())
        td.Style.Add("width", widthPercent.ToString() + "%")
        td.Controls.Add(New LiteralControl(tabPage.HeaderText))
        tr.Controls.Add(td)
    Next
    Dim tdSpace As HtmlTableCell = New HtmlTableCell
    tdSpace.Controls.Add(New LiteralControl("&nbsp;"))
    tdSpace.Attributes.Add("class", "TabViewHorizontal_Space")
    tr.Controls.Add(tdSpace)
    table.Controls.Add(tr)

    ' tabPage View
    For Each tabPage In Me.TabPages
        Dim displayStyle As String = "none"
        If (Me.SelectedIndex = Me.TabPages.IndexOf(tabPage)) Then
            displayStyle = "block"
        End If

        If (Me.SelectedIndex = Me.TabPages.IndexOf(tabPage)) Then
            tr = New HtmlTableRow
            Dim td As HtmlTableCell = New HtmlTableCell
            td.ColSpan = Me.TabPages.Count + 1
            td.Attributes.Add("class", "TabViewHorizontal_Content")
            If (tabPage.HasControls() = False) Then
                tabPage.Controls.Add(New LiteralControl("&nbsp;"))
            End If
            td.Controls.Add(tabPage)
            tr.Controls.Add(td)
            tr.Style.Add("display", displayStyle)
            table.Controls.Add(tr)
        End If
    Next
    table.RenderControl(output)
End Sub

```

If you compile this code and place the control on an ASP.NET Web page, you should see a tabbed control with the first tab always selected when you view the page in the browser. You need to add support for tab selection by defining a *SelectedIndexChanged* event and by enabling the control to support postbacks. The basic idea is that when a user clicks a tab header, the current view should change to correspond to the tab header clicked. To be consistent with other ASP.NET server controls, you should give this control the capability to support both client-side selection changes and server-side selection changes. Do this by providing a *Boolean AutoPostBack* property, as shown here:

```
Private _autoPostBack As Boolean = True
```

```
...
```

```
Public Property AutoPostBack() As Boolean
    Get
        Return _autoPostBack
    End Get
    Set(ByVal Value As Boolean)
        _autoPostBack = Value
    End Set
End Property
```

For server-side selection changes, you only need to re-render the control with the selected view shown. For client-side changes, however, things will get a little more difficult. On the client, the entire tabbed control should be rendered with the inactive views hidden. Therefore, you must add client-side script to activate the hidden views as the tab pages are changed. Like the style sheet file, add a *TabView.htc* script file to the project and set its Build Action property to Embedded Resource.

Add this code to the script file:

```
<public:property name="orientation" />
<public:attach event="onclick" onevent="TabView_OnClick()" />
<script language="jscript">
function TabView_OnClick()
```

```
{
  if (orientation == "horizontal")
  {
    TabViewHorizontal_OnClick();
  }
  else if (orientation == "vertical")
  {
    TabViewVertical_OnClick();
  }
}

function TabViewHorizontal_OnClick()
{
  var nTabPageHeaderHit = 0;
  var tdTabPageHeaderHit = event.srcElement;
  var trHeader = tdTabPageHeaderHit.parentNode;
  var tdTabs = trHeader.childNodes;
  for (var nTab = 0; nTab < tdTabs.length - 1; nTab = nTab + 1)
  {
    var tdTabHeader = tdTabs.item(nTab);
    tdTabHeader.className = "TabViewHorizontal_Header";
    if (tdTabHeader == tdTabPageHeaderHit)
    {
      nTabPageHeaderHit = nTab;
      document.cookie = trHeader.parentNode.parentNode.id +
"_SelectedIndex=" + nTab;
    }
    tdTabPageHeaderHit.className = "TabViewHorizontal_HeaderSelected";

    var tBody = trHeader.parentNode;
    var trTabPageViews = tBody.childNodes;
    if ( trTabPageViews.length <= 1)
      return;
    for (var nTabView = 1; nTabView < trTabPageViews.length; nTabView =
nTabView + 1)
    {
      var trTabPageView = trTabPageViews.item(nTabView);
      if (nTabPageHeaderHit == nTabView - 1)
      {
        trTabPageView.style.display = "block";
      }
      else
      {
        trTabPageView.style.display = "none";
      }
    }
  }
}

function TabViewVertical_OnClick()
{
  var tdTabPageHeaderHit = event.srcElement;
```

```

var tBody = tdTabPageHeaderHit.parentNode.parentNode;
var trTabs = tBody.childNodes;
for (var nTab = 0; nTab < trTabs.length; nTab = nTab + 1)
{
    if ( ( nTab % 2) == 0)
    {
        var tdTabHeader = trTabs.item(nTab).firstChild;
        tdTabHeader.className = "TabViewVertical_Header";
        if (tdTabHeader == tdTabPageHeaderHit)
        {
            document.cookie = tBody.parentNode.id + "_SelectedIndex=" +
nTab/2;
        }
    }
    else
    {
        var trTabPageView = trTabs.item(nTab);
        trTabPageView.style.display = "none";
    }
}
var trTabPageView = tdTabPageHeaderHit.parentNode.nextSibling;
trTabPageView.style.display = "block";
tdTabPageHeaderHit.className = "TabViewVertical_HeaderSelected";
}
</script>

```

Because this script is a behavior script, you need to modify the style sheet file to make it work, as shown here:

```

.TabViewHorizontal_Header
{
    behavior:url(TabView.htc);
    ...
}
.TabViewHorizontal_HeaderSelected
{
    behavior:url(TabView.htc);
    ...
}
.TabViewVertical_Header
{
    behavior:url(TabView.htc);
    ...
}
.TabViewVertical_HeaderSelected
{
    behavior:url(TabView.htc);
    ...
}

```

```
}

```

Now declare an event named *SelectedIndexChanged* and a corresponding virtual method named *OnSelectedIndexChanged*, as shown here:

```
Public Event SelectedIndexChanged As EventHandler

...

Public Overridable Sub OnSelectedIndexChanged(ByVal e As EventArgs)
    RaiseEvent SelectedIndexChanged(Me, e)
End Sub

```

To allow the control to support server-side post backs, you need to implement the *IPostBackDataHandler* and *IPostBackEventHandler* interfaces. You also need to register the control as one that requires postback processing if the *AutoPostBack* property is set to *True*.

Here is the updated class:

```
Public Class TabView
    Inherits Control
    Implements INamingContainer
    Implements IPostBackDataHandler
    Implements IPostBackEventHandler

...

Protected Overrides Sub OnInit(ByVal e As EventArgs)
    MyBase.OnInit(e)
    If (AutoPostBack AndAlso Not Page Is Nothing) Then
        Page.RegisterRequiresPostBack(Me)
    End If
End Sub

Sub RaisePostDataChangedEvent() Implements
IPostBackDataHandler.RaisePostDataChangedEvent
    ' Stub Implementation, Required for IPostBackDataHandler
    ' This control must implement IPostBackDataHandler, even though it
    doesn't use its methods.
End Sub

Function LoadPostData(ByVal strPostDataKey As String, ByVal
postDataCollection As System.Collections.Specialized.NameValueCollection)
As Boolean Implements IPostBackDataHandler.LoadPostData

```

```

' Stub Implementation, Required for IPostBackDataHandler
' This control must implement IPostBackDataHandler, even though it
doesn't use its methods.
Return False
End Function

Sub RaisePostBackEvent(ByVal eventArgument As String) Implements
IPostBackEventHandler.RaisePostBackEvent
    If (eventArgument Is Nothing) Then
        Return
    End If

    SelectedIndex = Int32.Parse(eventArgument)

    Dim e As EventArgs = New EventArgs
    OnSelectedIndexChanged(e)
End Sub

```

This control implements the *IPostBackDataHandler* interface because it registers the page as one that requires postback. Thus, the page framework will expect this interface to be implemented by the control, even though the control only provides a stubbed implementation of this interface. This control implements the *IPostBackEventHandler* interface so that it can process the postback events raised by the client. The *RaisePostBackEvent* method of this interface is implemented to set the *SelectedIndex* property using an argument that was passed from the client. This method calls the *OnSelectedIndexChanged* method after the *SelectedIndex* property is set so that the *SelectedIndexChanged* event gets raised.

As stated earlier, you only need to re-render the control with the selected view shown for server-side changes. For client-side changes, the entire control should be rendered with the inactive views hidden. The following methods are updated to accommodate this:

```

Protected Overrides Sub Render(ByVal output As
System.Web.UI.HtmlTextWriter)
    If (Not Page Is Nothing AndAlso Not
Page.IsClientScriptBlockRegistered(Me.GetType().FullName)) Then
        Page.RegisterClientScriptBlock(Me.GetType().FullName, "")
        RenderSupportScripts(output)
    End If

    If (TabPage.Count > 0) Then

```

```

    If (Not AutoPostBack AndAlso Not Page Is Nothing) Then
        SelectedIndex = 0
        Dim cookie As HttpCookie = Page.Request.Cookies(Me.UniqueID +
            "_SelectedIndex")
        If (Not cookie Is Nothing) Then
            SelectedIndex = Int32.Parse(cookie.Value)
        End If
        If (SelectedIndex < 0 OrElse SelectedIndex > TabPages.Count -
1) Then
            SelectedIndex = 0
        End If
    End If
    If (Orientation = TabViewOrientation.Vertical) Then
        RenderControlVertical(output)
    ElseIf (Orientation = TabViewOrientation.Horizontal) Then
        RenderControlHorizontal(output)
    End If
End If
End Sub

Private Sub RenderControlVertical(ByVal output As HtmlTextWriter)
    ' TabView
    Dim table As HtmlTable = New HtmlTable
    table.ID = Me.UniqueID
    table.Width = Me.Width.ToString()
    table.Height = Me.Height.ToString()
    table.CellSpacing = 1
    table.CellPadding = 0
    table.Border = 1
    table.BgColor = "Menu"
    table.Style.Add("cursor", "default")

    ' Render each tab page in the TabPageCollection.
    Dim tabPage As TabPage
    For Each tabPage In Me.TabPages
        Dim displayStyle As String = "none"
        If (Me.SelectedIndex = Me.TabPages.IndexOf(tabPage)) Then
            displayStyle = "block"
        End If

        ' TabPage Header
        Dim tr As HtmlTableRow = New HtmlTableRow
        Dim td As HtmlTableCell = New HtmlTableCell
        If (Me.SelectedIndex = Me.TabPages.IndexOf(tabPage)) Then
            td.Attributes.Add("class", "TabViewVertical_HeaderSelected")
        Else
            td.Attributes.Add("class", "TabViewVertical_Header")
        End If
        If (AutoPostBack) Then
            td.Attributes.Add("onclick", "jscript:" +
Page.GetPostBackEventReference(Me, TabPages.IndexOf(tabPage).ToString())
        End If
    End For
End Sub

```

```

        td.Attributes.Add("orientation",
Me.Orientation.ToString().ToLower())
        td.Controls.Add(New LiteralControl(tabPage.HeaderText))
        tr.Controls.Add(td)
        table.Controls.Add(tr)

' tabPage View
    If (Not Me.AutoPostBack OrElse (Me.AutoPostBack AndAlso
Me.SelectedIndex = Me.TabPages.IndexOf(tabPage))) Then
        tr = New HtmlTableRow
        td = New HtmlTableCell
        td.Attributes.Add("class", "TabViewVertical_Content")
        td.Controls.Add(tabPage)
        tr.Controls.Add(td)
        tr.Style.Add("display", displayStyle)
        table.Controls.Add(tr)
    End If
Next

    table.RenderControl(output)
End Sub

Private Sub RenderControlHorizontal(ByVal output As HtmlTextWriter)
' TabView
    Dim table As HtmlTable = New HtmlTable
    Table.ID = Me.UniqueID
    Table.Width = Me.Width.ToString()
    Table.Height = Me.Height.ToString()
    Table.CellSpacing = 0
    Table.CellPadding = 0
    Table.Border = 0
    Table.BgColor = "Menu"
    Table.Style.Add("cursor", "default")

    Dim tr As HtmlTableRow = New HtmlTableRow

    Dim widthPercent As Integer = 15
    If (Me.TabPages.Count * widthPercent > 100) Then
        widthPercent = 100 / Me.TabPages.Count
    End If

' Render each tab page in the tabPageCollection.
    Dim tabPage As tabPage
    For Each tabPage In Me.TabPages
        ' tabPage Header
        Dim td As HtmlTableCell = New HtmlTableCell
        If (Me.SelectedIndex = Me.TabPages.IndexOf(tabPage)) Then
            td.Attributes.Add("class", "TabViewHorizontal_HeaderSelected")
        Else
            td.Attributes.Add("class", "TabViewHorizontal_Header")
        End If
        If (AutoPostBack) Then

```

```

        td.Attributes.Add("onclick", "jscript:" +
Page.GetPostBackEventReference(Me, TabPages.IndexOf(tabPage).ToString())
    End If
    td.Attributes.Add("orientation",
Me.Orientation.ToString().ToLower())
    td.Style.Add("width", widthPercent.ToString() + "%")
    td.Controls.Add(New LiteralControl(tabPage.HeaderText))
    tr.Controls.Add(td)
Next
Dim tdSpace As HtmlTableCell = New HtmlTableCell
tdSpace.Controls.Add(New LiteralControl("&nbsp;"))
tdSpace.Attributes.Add("class", "TabViewHorizontal_Space")
tr.Controls.Add(tdSpace)
table.Controls.Add(tr)

' tabPage View
For Each tabPage In Me.TabPages
    Dim displayStyle As String = "none"
    If (Me.SelectedIndex = Me.TabPages.IndexOf(tabPage)) Then
        displayStyle = "block"
    End If

    If (Not Me.AutoPostBack OrElse (Me.AutoPostBack AndAlso
Me.SelectedIndex = Me.TabPages.IndexOf(tabPage))) Then
        tr = New HtmlTableRow
        Dim td As HtmlTableCell = New HtmlTableCell
        td.ColSpan = Me.TabPages.Count + 1
        td.Attributes.Add("class", "TabViewHorizontal_Content")
        If (tabPage.HasControls() = False) Then
            tabPage.Controls.Add(New LiteralControl("&nbsp;"))
        End If
        td.Controls.Add(tabPage)
        tr.Controls.Add(td)
        tr.Style.Add("display", displayStyle)
        table.Controls.Add(tr)
    End If
Next
table.RenderControl(output)
End Sub

```

If the *AutoPostBack* property is set to *True*, the code uses the *GetPostBackEventReference* method of the *Page* class to set the *onclick* event of the tab page headers. Otherwise, the *onclick* event defined in the script file is used. This completes the runtime implementation for the *TabView* control.

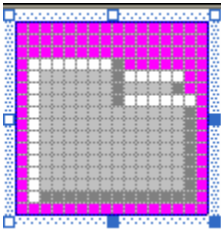
Step 4: Design-Time Support

For this control to be developer-friendly in the Visual Studio .NET 2003 IDE, you should add some design-time support to it. One of the first things to do is to decorate the control with design-time attributes, such as *DefaultValueAttribute*, *DefaultEventAttribute*, and *DefaultPropertyAttribute* objects. Here are some attributes that are applied to the control:

```
<DefaultProperty("TabPage"), _
DefaultEvent("SelectedIndexChanged"), _
ToolboxBitmap(GetType(TabView), "TabView.bmp"), _
Designer(GetType(TabViewDesigner)), _
ToolboxData("<0:TabView runat=server></0:TabView>"), _
PersistChildren(False), _
ParseChildren(True, "TabPage")> _
Public Class TabView
```

The *DefaultPropertyAttribute* object indicates that the *TabPage* property should immediately gain focus in the property browser when the *TabView* control is selected on the designer surface. The *DefaultEventAttribute* object indicates that an event handler for the *SelectedIndexChanged* event should be created automatically if a user double-clicks the *TabView* control in design mode. The *ToolboxBitmapAttribute* class is also used to indicate that the bitmap that should be used for the *TabView* control exists as an embedded resource named *TabView.bmp*. And last but not least, the *DesignerAttribute* object specifies the designer that should be used to render the *TabView* control in design mode.

Create a new folder named *Design* in the project and add a new bitmap named *TabView.bmp* to this folder. Set the file's Build Action property to *Embedded Resource*. Here is the bitmap that I created:



Because a *DesignerAttribute* object was specified, you need to go ahead and implement a *TabViewDesigner* class that derives from *ControlDesigner*. The first thing to do is provide rendering logic so that the designer knows how to render the control in design mode. You can't use the base implementation since it relies on the control to render itself, and the control's rendering logic extracts scripts and styles from the resources, which can't be accomplished easily at design time. Therefore, you need to override the *GetDesignTimeHtml* and *GetEmptyDesignTimeHtml* methods, as shown here:

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

Public Class TabViewDesigner
    Inherits ControlDesigner

    Private _designTimeSelectedIndex As Integer

    Protected Overrides Function GetEmptyDesignTimeHtml() As String
        Dim html As String = ""
        html = (html & "<table style='font-family: Tahoma; font-size: 8pt;
color:buttontext; background-color:buttonface; border: solid 1px border-
top-color: buttonhighlight; border-left-color: buttonhighlight; border-
right-color: buttonshadow; border-bottom-color: buttonshadow'>")
        html = (html & "<tr><td><b>TabView</b> - " & Me.ID & "</td></tr>")
        html = (html & "<tr><td>Please add tab pages through the TabPages
(Collection) property in the property browser,</td></tr>")
        html = (html & "<tr><td>or by right-clicking this control and
choosing 'Add New Tab Page'.</td></tr>")
        Return (html & "<tr><td>Then switch to HTML view and edit each tab
page's view by inserting inner content.</td></tr></table>")
    End Function

    Public Overrides Function GetDesignTimeHtml() As String
        Dim tabView As TabView = MyBase.Component
        If ((tabView Is Nothing) OrElse (tabView.TabPages.Count <= 0)) Then
            Return Me.GetEmptyDesignTimeHtml
        End If
    End Function
End Class
```

```

End If
Dim stringWriter As New StringWriter
Dim textWriter As New HtmlTextWriter(stringWriter)
If (tabView.Orientation = tabView.TabViewOrientation.Horizontal)
Then
    ' TabView
    Dim table As New HtmlTable
    table.Width = tabView.Width.ToString
    table.Height = tabView.Height.ToString
    table.CellSpacing = 0
    table.CellPadding = 0
    table.Border = 0
    table.BgColor = "Menu"

    Dim tr As New HtmlTableRow

    Dim widthPercent As Integer = 15
    If ((tabView.TabPages.Count * widthPercent) > 100) Then
        widthPercent = (100 / tabView.TabPages.Count)
    End If

    ' Render each tabpage in the TabPageCollection
    Dim tabPage As TabPage
    For Each tabPage In tabView.TabPages
        ' TabPageHeader
        Dim td As New HtmlTableCell
        If (Me._designTimeSelectedIndex =
tabView.TabPages.IndexOf(tabPage)) Then
            ' TabPage Header Selected Styles
            td.Style.Add("border-top", "thin outset white")
            td.Style.Add("border-right", "thin outset")
            td.Style.Add("border-left", "thin outset white")
            td.Style.Add("padding-top", "0px")
            td.Style.Add("padding-bottom", "0px")
            td.Style.Add("padding-left", "5px")
            td.Style.Add("padding-right", "0px")
            td.Style.Add("margin", "0px")
            td.Style.Add("height", "20px")
            td.Style.Add("width", (widthPercent.ToString & "%;"))
            td.Style.Add("font-size", "smaller")
            td.Style.Add("font-weight", "bold")
            td.Style.Add("font-family", "Menu, Verdana, Arial")
            td.Style.Add("color", "menutext")
            td.Style.Add("background-color", "menu")
        Else
            ' TabPage Header Default Styles
            td.Style.Add("border-top", "thin outset white")
            td.Style.Add("border-bottom", "thin inset window")
            td.Style.Add("border-right", "thin outset white")
            td.Style.Add("border-left", "thin outset white")
            td.Style.Add("padding-top", "0px")
            td.Style.Add("padding-bottom", "0px")
        End If
    Next
End For
End If

```

```

        td.Style.Add("padding-left", "5px")
        td.Style.Add("padding-right", "0px")
        td.Style.Add("margin", "0px")
        td.Style.Add("height", "20px")
        td.Style.Add("width", (widthPercent.ToString & "%;"))
        td.Style.Add("font-size", "smaller")
        td.Style.Add("font-weight", "400")
        td.Style.Add("font-family", "Menu, Verdana, Arial")
        td.Style.Add("color", "menutext")
        td.Style.Add("background-color", "eeeeee")
    End If
    td.Controls.Add(New LiteralControl(tabPage.HeaderText))
    tr.Controls.Add(td)
Next
Dim tdSpace As New HtmlTableCell
tdSpace.Controls.Add(New LiteralControl("&nbsp;"))
tdSpace.Style.Add("border-bottom", "thin inset window")
Dim width As Integer = (100 - (tabView.TabPages.Count * 15))
tdSpace.Style.Add("width", (width.ToString & "%;"))
tdSpace.Style.Add("font", "8pt verdana")
tdSpace.Style.Add("padding", "0,0,0,0")
tdSpace.Style.Add("background-color", "menu")
tr.Controls.Add(tdSpace)
table.Controls.Add(tr)

' tabPageView
For Each tabPage In tabView.TabPages
    Dim displayStyle As String = "none"
    If (Me._designTimeSelectedIndex =
tabView.TabPages.IndexOf(tabPage)) Then
        displayStyle = "block"
    End If
    tr = New HtmlTableRow
    Dim td As New HtmlTableCell
    td.ColSpan = (tabView.TabPages.Count + 1)
    td.Style.Add("border-top", "medium-none")
    td.Style.Add("border-bottom", "thin outset white")
    td.Style.Add("border-left", "thin outset white")
    td.Style.Add("border-right", "thin outset white")
    td.Style.Add("padding", "5px")
    td.Style.Add("padding-bottom", "5px")
    td.Style.Add("padding-left", "5px")
    td.Style.Add("padding-right", "5px")
    td.Style.Add("margin", "0px")
    td.Style.Add("height", "100%")
    td.Style.Add("width", "100%")
    td.Style.Add("background-color", "menu")
    td.Style.Add("cursor", "auto")
    If Not tabPage.HasControls Then
        tabPage.Controls.Add(New LiteralControl("&nbsp;"))
    End If
    td.Controls.Add(tabPage)

```

```

        tr.Controls.Add(td)
        tr.Style.Add("display", displayStyle)
        table.Controls.Add(tr)
    Next
    table.RenderControl(textWriter)
Else
    If (tabView.Orientation = tabView.TabViewOrientation.Vertical)
Then
    ' TODO: Implement
    stringWriter.Write(MyBase.GetDesignTimeHtml)
    End If
    End If
    Return stringWriter.ToString
End Function
End Class

```

The *GetEmptyDesignTimeHtml* method simply informs the user of how to add tab pages. The *GetDesignTimeHtml* method calls the *GetEmptyDesignTimeHtml* method if no tab pages have been added to the TabView control. Otherwise, the *GetDesignTimeHtml* method renders the TabView control in the same way that the TabView control renders itself at run time. The only exception is that the *GetDesignTimeHtml* method manually renders the styles that were added to the TabView.css file.

Note that a variable named *_designTimeSelectedIndex* is used to control the rendering logic. This variable is used so that the tab pages can be cycled in design mode without affecting the “true” *SelectedIndex* property of the TabView control. To support cycling, as well as adding and removing tab pages, you need to implement designer verbs. Here is the modified class:

```

Private _verbAddTabPage As DesignerVerb
Private _verbShowNextTabPage As DesignerVerb
Private _verbRemoveTabPage As DesignerVerb

Public Sub New()
    Me._verbAddTabPage = New DesignerVerb("Add Tab Page", New
EventHandler(AddressOf Me.AddTabPage))
    Me._verbShowNextTabPage = New DesignerVerb("Show Next Page", New
EventHandler(AddressOf Me.ShowNextTabPage))
    Me._verbRemoveTabPage = New DesignerVerb("Remove Current Page", New
EventHandler(AddressOf Me.RemoveTabPage))
    MyBase.Verbs.Add(Me._verbAddTabPage)

```

```

MyBase.Verbs.Add(Me._verbShowNextTabPage)
MyBase.Verbs.Add(Me._verbRemoveTabPage)
End Sub

Public Overrides Sub Initialize(ByVal component As IComponent)
MyBase.Initialize(component)
Dim tabView As TabView = CType(component, TabView)
Me._designTimeSelectedIndex = tabView.SelectedIndex
End Sub

Private Sub AddTabPage(ByVal sender As Object, ByVal e As EventArgs)
Dim tabView As TabView = CType(MyBase.Component, TabView)
Dim tabPage As New TabPage
tabView.TabPages.Add(tabPage)
tabView.Controls.Add(tabPage)
tabPage.HeaderText = "Tab"
tabPage.ID = tabPage.ClientID
Me._designTimeSelectedIndex = (tabView.TabPages.Count - 1)
Me.OnComponentChanged(Me, New ComponentChangedEventArgs(tabView,
Nothing, Nothing, Nothing))
End Sub

Private Sub ShowNextTabPage(ByVal sender As Object, ByVal e As
EventArgs)
Dim tabView As TabView = CType(MyBase.Component, TabView)
Dim tabPageCount As Integer = tabView.TabPages.Count
If (tabPageCount > 0) Then
Me._designTimeSelectedIndex = ((Me._designTimeSelectedIndex + 1)
Mod tabPageCount)
Me.UpdateDesignTimeHtml()
End If
End Sub

Private Sub RemoveTabPage(ByVal sender As Object, ByVal e As
EventArgs)
Dim tabView As TabView = MyBase.Component
If (tabView.TabPages.Count <= 0) Then
Return
End If

tabView.TabPages.RemoveAt(Me._designTimeSelectedIndex)

' Ensure the selected index is still valid.
' It is possible that the selected Tab was the last one, and that
one was removed.
If (tabView.SelectedIndex >= tabView.TabPages.Count) Then
tabView.SelectedIndex = (tabView.TabPages.Count - 1)
If (tabView.SelectedIndex < 0) Then
tabView.SelectedIndex = 0
End If
End If

```

```

    ' Commit the change
    Me.OnComponentChanged(Me, New ComponentChangedEventArgs(tabView,
Nothing, Nothing, Nothing))

    ' Show the closest, right-most sibling Tab in the Designer
    Me._designTimeSelectedIndex -= 1
    If ((Me._designTimeSelectedIndex + 1) >= tabView.TabPages.Count)
Then
        Me._designTimeSelectedIndex -= 1
    End If
    Me.ShowNextTabPage(Me, EventArgs.Empty)
End Sub

```

Add an *Imports* statement for the *System.ComponentModel.Design* namespace at the beginning of the source file so that the *DesignerVerb* class can be recognized by the compiler. Finally, you need to modify the verbs so that they are displayed only when appropriate. For example, the verb associated with the removal of tab pages should not be visible when no tab pages exist. To do this, you need to obtain a reference to an *ISelectionService* object and attach an event handler to its *SelectionChanged* event. Then, whenever the event handler is called, you need to modify the visibility of the verbs. You also need to override the *OnComponentChanged* method and modify the visibility of the verbs. Here is the code:

```

Private _selectionService As ISelectionService

Public Overrides Sub Initialize(ByVal component As IComponent)
    MyBase.Initialize(component)
    Dim tabView As TabView = CType(component, TabView)
    Me._designTimeSelectedIndex = tabView.SelectedIndex
    If (Not _selectionService Is Nothing) Then
        AddHandler _selectionService.SelectionChanged, New
EventHandler(AddressOf OnSelectionChanged)
    End If
End Sub

Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    Me._selectionService = Me.GetService(GetType(ISelectionService))
    If (Not Me._selectionService Is Nothing) Then
        RemoveHandler Me._selectionService.SelectionChanged, New
EventHandler(AddressOf Me.OnSelectionChanged)
    End If
    MyBase.Dispose(disposing)
End Sub

```

```
Protected Overridable Sub OnSelectionChanged(ByVal sender As Object,
ByVal e As EventArgs)
    If ((Not Me._selectionService Is Nothing) AndAlso
Me._selectionService.GetComponentSelected(MyBase.Component)) Then
        Me.ModifyMenu()
    End If
End Sub

Public Overrides Sub OnComponentChanged(ByVal sender As Object, ByVal
ce As ComponentChangedEventArgs)
    MyBase.OnComponentChanged(sender, ce)
    If (ce.Component Is MyBase.Component) Then
        Me.ModifyMenu()
    End If
End Sub

Private Sub ModifyMenu()
    Dim tabView As TabView = MyBase.Component
    Me._verbShowNextTabPage.Visible = False
    Me._verbRemoveTabPage.Visible = False
    Me._verbShowNextTabPage.Visible = (tabView.TabPages.Count > 1)
    Me._verbRemoveTabPage.Visible = (tabView.TabPages.Count > 0)
End Sub
```

The previous code updates the verbs when either the selection has changed or the component has changed. It enables the “Show Next Page” verb only when there are two or more tab pages. It enables the “Remove Current Page” verb only when there are one or more tab pages.

Summary

You have now completed one of the most useful Web server controls there is. This chapter focused on some topics that were introduced earlier in the book, such as state management, rendering, parsing, and persistence. You learned how easy it is to implement design-time rendering so that the output looks exactly like an actual runtime control. With the knowledge from this and previous chapters, you should feel comfortable creating your own Web server controls and designers.