

Populating the TreeView Control from a Database

By Don Schlichting

Introduction

TreeView is a new Internet Explorer WebControl for ASP.NET that presents hierarchical datasets, folders, and XML documents in a familiar parent child view similar to the Windows File Explorer. This article will provide an introduction to the TreeView and then provide detailed steps for populating the tree from a database.

About the TreeView

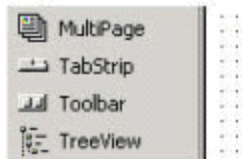
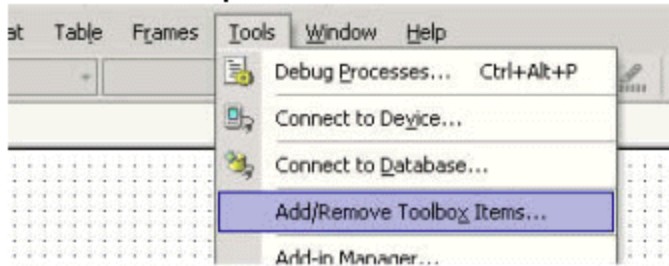
Word of WARNING. Developer Beware. Microsoft does not support any WebControl, including the TreeView. They do, however, host a newsgroup and a forum. The newsgroup is located on the public news server: "news.Microsoft.com". The group name is "microsoft.public.dotnet.framework.aspnet.webcontrols". The forum can be found at <http://www.asp.net/Forums/ShowForum.aspx?tabindex=1&ForumID=91>. Again, Microsoft does not provide any level of guaranteed support for either of these.

The TreeView is only one item in the Internet Explorer WebControls download set. The WebControl set also includes the MultiPage WebControl (for creating collections of page areas), a TabStrip WebControl (for authoring tabbed menus and navigation systems), and the ToolBar WebControl (used to create tool bar strips of control buttons). These tools give ASP.NET developers the ability to create Web-based applications that look, feel, and behave like traditional Windows desktop applications. All the controls include automatic browser detection so your pages will render and function in all commonly used browsers. For Internet Explorer 5.5 and above, the controls include DHTML behaviors for client-side functionality.

The Internet Explorer WebControls are not part of the .NET Framework and must be downloaded separately.

Downloading and Installing the WebControls Set

1. Download the Control from Microsoft at:
http://msdn.microsoft.com/downloads/samples/internet/asp_dot_net_servercontrols/webcontrols/default.asp
. Save the exe file.
2. Double click your saved IEWebControls.exe file to begin a typical InstallShield setup. You must have the .NET Framework loaded on your machine prior to installing the WebControls.
3. **Browse to your C:\Program Files\IE Web Controls folder and open the file "build.bat" in an editor. Modify the line with "csc.exe" in it to include the full path to your DOT.NET Framework, for example:**
"c:\winnt\microsoft.net\framework\v1.1.4322\csc.exe"
4. Double click or execute the build.bat file we just modified.
5. Check the newly created "C:\Program Files\IE Web Controls\Build" folder for the file "Microsoft.Web.UI.WebControls.dll". If it doesn't exist, double check step three.
6. Follow the remaining steps in the C:\Program Files\IE Web Controls\Readme.txt file for directions on creating and copying file to your IIS directory.
7. For Visual Studio users: Add the WebControls to the Visual Studio toolbox by selecting Tools, Add/Remove Tool Box Items from the menu. Then browse to the location of your "Microsoft.Web.UI.WebControls.dll".



After you're done, TreeView should be listed as a .NET Component with an icon for the TreeView.

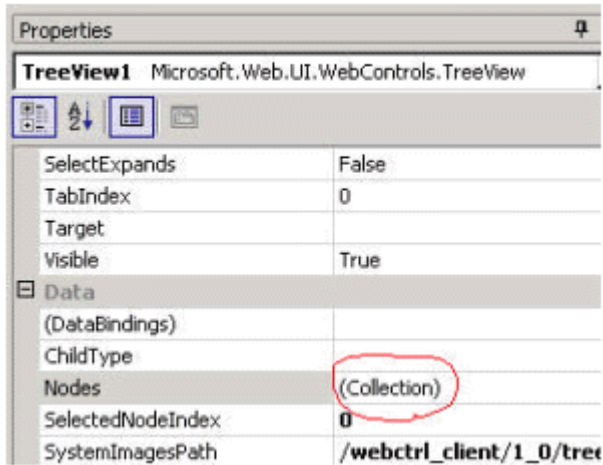
Creating a Simple TreeView

Lets start by creating the "Hello World" version of a TreeView. If you are coding by hand, create this new file:

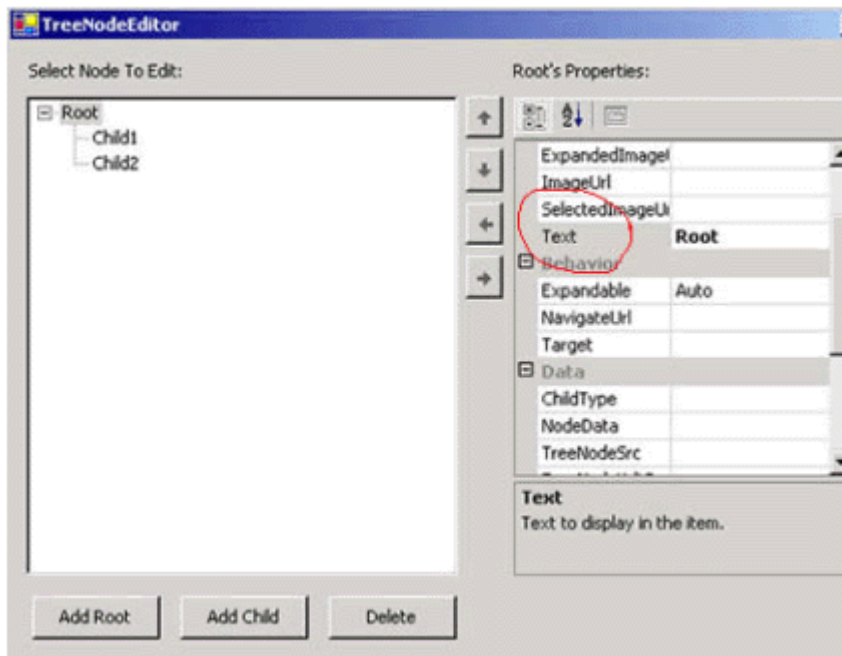
```
<%@ Register TagPrefix="iewc" Namespace="Microsoft.Web.UI.WebControls"
Assembly="Microsoft.Web.UI.WebControls" %>
<%@ Page Language="vb" AutoEventWireup="false" %>

<HTML>
  <HEAD>
    <title>Simple Tree</title>
  </HEAD>
  <body>
    <form id="Form1" method="post" runat="server">
      <iewc:TreeView id="TreeView1" runat="server">
        <iewc:TreeNode Text="Parent">
          <iewc:TreeNode
Text="Child1"></iewc:TreeNode>
          <iewc:TreeNode
Text="Child2"></iewc:TreeNode>
        </iewc:TreeNode>
      </iewc:TreeView>
    </form>
  </body>
</HTML>
```

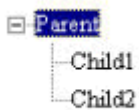
For Visual Studio users, drag the TreeView icon on your page. With TreeView selected in the Properties area, click the Nodes (Collection).



Next, add a root and two children. Change the text properties for each in the TreeNodeEditor.



This should give you the following Web page display



Because the TreeView is not part of the .NET Framework, lines one and two import and register the WebControls namespace.

Checkbox

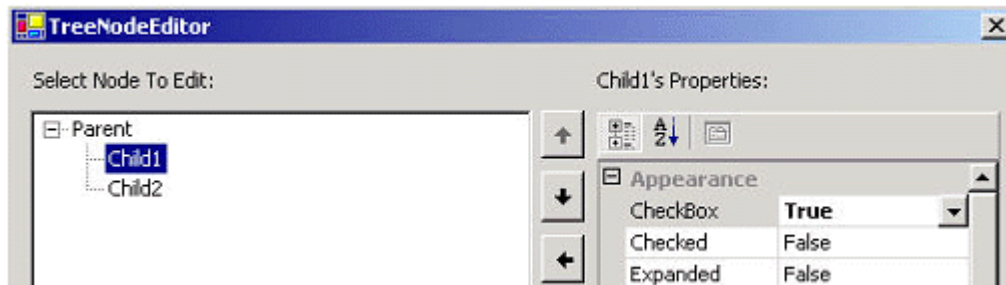
We can generate two very different feels for the control by using the "Checkbox" property.



Checkboxes are controlled at the node level by including "checkbox true" into the node line:

```
<iewc:TreeNode CheckBox="True" Text="Child1"></iewc:TreeNode>
```

In Visual Studio, the checkbox property is found in the Appearance area in the TreeNodeEditor.



XML

Before going into database binding, we'll look at XML because it's probably the most common way to bind data to a TreeView. This time, we will use the TreeNodeSrc attribute and static XML file to create the Parent Child structure of our tree.

Create the following XML file called XMLFile1.xml and save it in the same directory as your TreeView aspx file.

```
<TREENODES>
<treenode text="Partent1">
<treenode text="Child-1a"/>
<treenode text="Child-1b"/>
</treenode>
<treenode text="Partent2">
<treenode text="Child-2a"/>
<treenode text="Child-2b"/>
</treenode>
</TREENODES>
```

The XML root element must be upper case and named <TREENODES> for the bind to work. If your XML will be coming from a source where you cannot control the root name and structure, an XLS sheet can be used to translate your formatting. The following error will be produced if the correct root is not used.

Server Error in '/Tree' Application.

*The XML loaded from
TreeNodeSrc=XMLFile1.xml,
TreeNodeXslSrc= did not contain the
required outer <TREENODES>
container.*

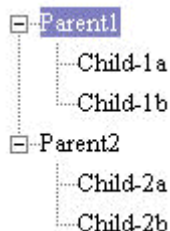
Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Exception: The XML loaded from
TreeNodeSrc=XMLFile1.xml, TreeNodeXslSrc= did not contain the required
outer <TREENODES> container.

Create the following aspx file and place it in the same directory as your XML file:

```
<%@ Register TagPrefix="iewc" Namespace="Microsoft.Web.UI.WebControls"
Assembly="Microsoft.Web.UI.WebControls" %>
<%@ Page Language="vb" %>
<HTML>
  <HEAD>
    <title>XMLIn</title>
  </HEAD>
  <body>
    <form id="Form1" method="post" runat="server">
  <iewc:TreeView id="TreeView1" runat="server
TreeNodeSrc="XMLFile1.xml"></iewc:TreeView>
    </form>
  </body>
</HTML>
```

These two files will produce the same structure we saw in our fist example.



The only difference from our first example is the `TreeNodeSrc="XMLFile1.xml"` attribute.

Database Populating

In many cases, you may want to populate your tree from dynamic data. Obtaining your values from a database allows menu and input trees to change on the fly. While there are ways to convert your database results into XML and then populate the tree with the converted data, we can skip this conversion step and fill the tree directly from database using straightforward ADO.NET.

For the database example, we'll generate a list of Suppliers and the Products each makes. The Northwind example database in SQL 2000 will be our source. Here's a view of the tree we'll create, collapsed and expanded:



On your Web form, drag a TreeView control over and name it TreeView1. Below is the control code. Remember to include an import for System.Data, System.Data.SqlClient, and Microsoft.Web.UI.WebControls:

```
Dim strConn As String = "server=.;database=Northwind;integrated security=true;"
```

```
Dim objConn As New SqlConnection(strConn)
```

```
Dim objDS As New DataSet
```

```
Dim daSuppliers As New SqlDataAdapter("SELECT CompanyName,SupplierID FROM  
Suppliers", objConn)
```

```
Dim daProducts As New SqlDataAdapter("SELECT ProductName, ProductID, SupplierID  
FROM Products", objConn)
```

```
daSuppliers.Fill(objDS, "dtSuppliers")
```

```
daProducts.Fill(objDS, "dtProducts")
```

```
objConn.Close()
```

```
objDS.Relations.Add("SuppToProd", _  
objDS.Tables("dtSuppliers").Columns("SupplierID"), _  
objDS.Tables("dtProducts").Columns("SupplierID"))
```

```
Dim nodeSupp, nodeProd As TreeNode
```

```
Dim rowSupp, rowProd As DataRow
```

```
For Each rowSupp In objDS.Tables("dtSuppliers").Rows
```

```
nodeSupp = New TreeNode
```

```
nodeSupp.Text = rowSupp("CompanyName")
```

```

nodeSupp.ID = rowSupp("SupplierID")
TreeView1.Nodes.Add(nodeSupp)
For Each rowProd In rowSupp.GetChildRows("SuppToProd")
    nodeProd = New TreeNode
    nodeProd.Text = rowProd("ProductName")
    nodeProd.ID = rowProd("ProductID")
    nodeSupp.Nodes.Add(nodeProd)
Next
Next

```

'clean up

```

objDS.Dispose()
daSuppliers.Dispose()
daProducts.Dispose()
objConn.Close()
objConn.Dispose()

```

Place the control code in your Page Load area. Again, remember to include an import for System.Data, System.Data.SqlClient, and Microsoft.Web.UI.WebControls.

The first three lines are standard database connection objects:

```

Dim strConn As String = "server=.;database=Northwind;integrated security=true;"
Dim objConn As New SqlConnection(strConn)
Dim objDS As New DataSet

```

The important point on the next two SQL statements is that each statement includes a common relation field. In this case, SupplierID will connect the Suppliers to their Products. We will create a separate Data Adapter for each parent child relationship. In this case, there are only two, Suppliers and Products. If there were a relation under Products, like Colors available, we would create a third Data Adapter that would include the ProductID so Colors could be tied back to Products:

```

Dim daSuppliers As New SqlDataAdapter("SELECT CompanyName,SupplierID FROM Suppliers",
objConn)

Dim daProducts As New SqlDataAdapter("SELECT ProductName, ProductID, SupplierID FROM
Products", objConn)

```

Next we fill both Data Adapters then close our connection:

```

daSuppliers.Fill(objDS, "dtSuppliers")
daProducts.Fill(objDS, "dtProducts")
objConn.Close()

```

Relation: The key to the TreeView is the relation. When a DataSet has more than one DataTable, a relation can be defined. There must be a common field in both to define the Parent Child relationship. In our case, it's the SupplierID.

```

objDS.Relations.Add("SuppToProd", _
    objDS.Tables("dtSuppliers").Columns("SupplierID"), _
    objDS.Tables("dtProducts").Columns("SupplierID"))

```

If we had our third relation of Color, there would be a second relation statement tying Products to Color.

Now we create our TreeView datatypes:

```
Dim nodeSupp, nodeProd As TreeNode
Dim rowSupp, rowProd As DataRow
```

Loop through our data sets and populate the TreeView. The ID attribute at +++ is not needed for our example to function. Including it gives a convenient method of determining your database ID values during click and check events.

```
For Each rowSupp In objDS.Tables("dtSuppliers").Rows
    nodeSupp = New TreeNode
    nodeSupp.Text = rowSupp("CompanyName")
    nodeSupp.ID = rowSupp("SupplierID") +++
    TreeView1.Nodes.Add(nodeSupp)
    For Each rowProd In rowSupp.GetChildRows("SuppToProd")
        nodeProd = New TreeNode
        nodeProd.Text = rowProd("ProductName")
        nodeProd.ID = rowProd("ProductID")
        nodeSupp.Nodes.Add(nodeProd)
        ***
    Next
Next
```

If you had a third relation, like the Colors available mentioned previously, we would create a third For Next loop located at the *** marker.

IndexChanged Event

In this last example, there are two labels included. They will tell us the old and new indexes when a change occurs. To get the event, include the AutoPostBack="True" in the TreeView's definition. Next, create a SelectIndexChanged Sub:

```
Private Sub TreeView1_SelectedIndexChanged(ByVal sender As Object, ByVal e As
Microsoft.Web.UI.WebControls.TreeViewSelectEventArgs) Handles
TreeView1.SelectedIndexChanged
    Label1.Text = e.OldNode.ToString
    Label2.Text = e.NewNode.ToString
End Sub
```

First, Bigfoot Breweries was clicked. Next Sasquatch Ale was selected, giving the following Old and New strings:



The Parent Child relation can be determined by the decimal. No decimal means a parent was selected, in our case, parent 1 (the TreeView is 0 based; Aux joyeux is 0, making Bigfoot 1). The child Ale is the first child (noted by number 0) of parent 1. If we had our third relation of Colors available under Sasquatch Ale, such as "Amber", "Dark", and Light", Amber would be 1.0.0, Dark would be 1.0.1, and Light would be 1.0.2.

Using the index together with the "nodeSupp.ID" will give you the table and ID of the item selected or checked.

There are also handy Expand and Collapse methods for controlling your tree's presentation.

Conclusion

The new TreeView WebControl is a quick way to give Web application a desktop look. The methods are easy and usually straightforward. Try to keep your trees small in size, or slow performance will be encountered. Custom images may be used rather than the standard Windows folder or checkbox picture. There are also several third party TreeView controls that offer additional methods and properties.