Using the TreeView Control
Doug Hennig
dhennig@stonefield.com

The new TreeView ActiveX control that comes with VFP 5.0 is a powerful, visually appealing tool you'll want to use in many applications. However, its quirks and poor documentation can make you spend many unproductive, frustrating hours figuring out how to properly use it. This article explores some helpful techniques for working with the TreeView control.

Visual FoxPro 5.0 includes many new ActiveX (formerly OLE) controls that can add great new features to your applications. Many of these, including the TreeView, ListView, StatusBar, and CommonDialog controls, allow you to create a Windows 95 interface for your applications. The ActiveX controls are easy to find: just select ActiveX Controls from the View Classes menu in the Form Control toolbar, and 30 new controls appear. Simply drag a control from the toolbar to a form as you would with any native control, give it a name, set some properties, and away you go. In addition to the VFP Properties sheet, each control has a custom Properties sheet. To access it, right-click on the control and select the appropriate item from the menu that appears.

To make things even better, you can take advantage of the fact that VFP is the only development environment that allows you to subclass ActiveX controls to add additional behavior. For example, you could create a wrapper class for a control to hide its complexity and simplify its use, or add custom properties and methods to further extend the controls.

The TreeView control
In my opinion, the most important new control is TreeView. You've already used the TreeView control -- it's used in the left pane of the Windows Explorer, and in the VFP Project Manager and Class Browser. This control is visually more appealing than the Outline control available in VFP 3.0, and is much easier to work with. For example, you had to add nodes to the Outline control in the order they appeared in the outline. With the TreeView control you can add nodes in any order because you can specify which node is the parent when you add a new one.

You can use the TreeView control to do many things. Here are some examples:

- Display a bill of materials list for an inventory control system.
- Use drill-downs. For example, you could display a list of customers and drill down to orders and products for each order. Another example is a list of regions, which expands to the salespeople under each region, each of which expands to their customers, and so on.
- Improve organization charts: Divisions are expanded to branches, which are expanded to supervisors, which are expanded to employees.
- Create any other hierarchical display of information.

The TreeView control is located in COMCTL32.OCX (in \WINDOWS\SYSTEM). If you've installed the Visual Basic Control Creation Edition (distributed at the 1996 Microsoft FoxPro Developers Conference and available from the Microsoft Web site), an updated version of COMCTL32.OCX -- which has a number of problems with VFP 5.0 -- was installed on your system. If your copy of COMCTL32.OCX is larger than 325K (the newer, problematic version is more than 500K), you'll want to reinstall the "good" copy from your VFP 5.0 CD-ROM (located in the \OS\SYSTEM directory on the CD).

The TreeView control is documented in CTRLREF.HLP (in the VFP home directory). The help information for the TreeView control is probably the poorest of all ActiveX control help (which is generally abysmal anyway): Its organization is confusing, the information is incorrect in several places, and it's frequently as clear as mud, especially in documenting how to access some methods and properties.

The TreeView control allows you to work with three different types of objects: the TreeView control itself, the Nodes collection, and Node objects. The Nodes collection is like the Controls collection of a form: it enables you to access individual node objects without going through the Nodes collection. For example, the TreeView SelectedItem property is an object reference to the selected node, and some methods - such as NodeClick and Expanded -- accept an object reference to a node as a parameter.

I'll explore the properties, events, and methods of these three types of objects separately. The accompanying Download file includes a sample form (TREEVIEW.SCX) that shows many of the techniques I'll discuss in this article.

TreeView control methods and events
The TreeView control responds to some of the methods and events we often associate with other VFP controls: Click, DblClick, Drag, DragDrop, DragOver, GotFocus, KeyDown, KeyPress, KeyUp, LostFocus, MouseDown, MouseMove,
MouseUp, Move, Refresh, SetFocus, ShowWhatsThis, and ZOrder. There's one big exception: there isn't a RightClick event. You'll see how to handle that later.

In addition to these methods and events, the TreeView control has some of its own methods and events (the help file lists Clear and Remove methods, but these really belong to the Nodes collection, not the TreeView control itself):

- **BeforeLabelEdit** and **AfterLabelEdit** occur before and after the label is edited by the user (as in Windows Explorer, you can click the selected node and edit its text, although you can disable this automatic editing capability as you'll see in a moment). Code in this event is usually used to save the new text somewhere, such as in a field in a table.

- **Collapse** and **Expand** are fired when users collapse or expand a node. They receive as a parameter an object reference to the selected node. Collapsing or expanding a node doesn't make it the active node, which is usually confusing to the user. Add the following code to the Collapse and Expand events to ensure that the node becomes active (the call to NodeClick(), a method I'll explain later, is only necessary if you have some custom code in that method):

```vbnet
lparameters node
Node.Selected = .T.
This.NodeClick(Node)
```

- **GetVisibleCount** shows the number of complete nodes visible in the control. This number might seem to be too low by one if the control can show most of the last node.

- **HitTest** returns a reference to a node object if one exists at the passed X and Y coordinates, or .NULL. if there isn't a node at those coordinates. One complication: HitTest expects the X and Y coordinates in a unit known as "twips" (used in Visual Basic), while VFP uses pixels. The following code will convert pixels to twips (thanks to Chin Bae and Mark Giesen for figuring this out). The TREEVIEW sample form has this code in its Init method, and has two custom properties to store the calculated values: nTreeFactorX and nTreeFactorY.

```vbnet
local liHWnd, ;
liHDC, ;
liPixelsPerInchX, ;
liPixelsPerInchY

* Define some constants.
#define cnLOG_PIXELS_X      88
* From WINGDI.H
#define cnLOG_PIXELS_Y      90
* From WINGDI.H
#define cnTWIPS_PER_INCH  1440
* 1440 twips per inch

* Declare some Windows API functions.
declare integer GetActiveWindow in WIN32API
declare integer GetDC           in WIN32API ;
integer iHDC
declare integer GetDeviceCaps   in WIN32API ;
integer iHDC, integer iIndex

* Get a device context for VFP.
liHWnd = GetActiveWindow()
liHDC  = GetDC(liHWnd)

* Get the pixels per inch.
liPixelsPerInchX = GetDeviceCaps(liHDC, cnLOG_PIXELS_X)
liPixelsPerInchY = GetDeviceCaps(liHDC, cnLOG_PIXELS_Y)

* Get the twips per pixel and store in properties of
* the form.

with This
  .nTreeFactorX = cnTWIPS_PER_INCH/liPixelsPerInchX
  .nTreeFactorY = cnTWIPS_PER_INCH/liPixelsPerInchY
endwith
```

HitTest() can help solve an issue with right-clicks of the mouse. You might want to display a pop-up menu when the user right-clicks on a node, but right-clicking on a node doesn't make it the selected node. Use the following code in the MouseDown event of the control to handle this (it's the only way you can handle right-clicks because there is no RightClick event). Again, the call to NodeClick() is necessary only if you have some code in that method:

```vbnet
lparameters Button, Shift, X, Y
local loNode
```
NodeClick is fired when the user clicks on a node (before the Click event). NodeClick receives as a parameter an object reference to the selected node object. Typically, this method is used to update some things (such as the values of other controls) when a node is selected. If the code in NodeClick takes too long to execute, the selected item will be highlighted but the former item will have a dashed line around it. Moving the mouse (even without clicking the mouse button) restores the highlight to the former item. Add the following code to the NodeClick event to ensure that the clicked node becomes selected:

StartLabelEdit is used to manually invoke the editing of a node’s label. This is used when the LabelEdit property is 1-Manual.

TreeView control properties
As with events, TreeView controls support some of the same properties supported by other controls, including DragIcon, DragMode, Enabled, and Visible. Many of the TreeView-specific properties can be set in the TreeView Control Property sheet invoked from the right-mouse menu. These include Style, LineStyle, Indentation, PathSeparator, and HideSelection.

Of the properties that can be edited in the property sheet, here are the ones you’re most likely to change from the defaults:

- **Style.** This property indicates what appears in the TreeView control. You can choose whether images appear, whether lines are displayed, whether the plus/minus marks are shown, and so on.
- **LineStyle.** This property indicates whether “root lines” are displayed. If you don’t set this to 1-Root Lines, you won’t get the plus/minus mark for the top-level objects, regardless of how Style is set.
- **LabelEdit.** If you don’t want the user to be able to edit the text for each node, set this to 1-Manual.
- **ImageList.** The images used by the nodes in the TreeView control aren’t specified with a filename such as the Picture property of a CommandButton. Instead, the images must be loaded into an ImageList control, another of the new ActiveX controls. The ImageList control is simple to use: just drop one on a form, bring up its property sheet from the right-click menu, and add images to it by clicking the Add button and locating the desired graphic files. The ImageList property of the TreeView control is a reference to the ImageList control containing the images used by the nodes in the TreeView. The drop-down list in the TreeView property sheet shows the name of existing ImageList controls on the form.
- **Indentation.** This property indicates how much each child node is indented.
- **HideSelection.** If you don’t turn this option off, the selected node doesn’t stay highlighted when the TreeView control loses focus. This can be very confusing to the user.
- **Font.** This property shows the font name, size, and style used for the node text.

The following properties aren’t available from the property sheet (the help file includes a ScrollBars property, but there’s no such property):

- **DropHighlight.** This is supposed to be used in the DragOver event of the TreeView control so you can highlight a node as the mouse passes over it. However, no one has been able to get this to work in VFP yet; setting this property to a node object gives a “type mismatch” error. I suspect it’s a documentation issue, but I’m not sure. You can get around this problem by using HitTest() to determine which node the mouse is over and setting that node’s Selected property to .T. The TREEVIEW sample form uses this technique in its DragOver event.
- **Nodes.** This property references the Nodes collection in the control.
- **SelectedItem.** This property references the currently selected node object.

**Nodes collection methods**
Here are the methods of the Nodes collection of the TreeView control (referenced by specifying <Object>.Nodes, where <Object> is the name of the TreeView control):

```vbp
if Button = 2
    * If this is the right mouse button, get a reference to the node under the mouse.
    loNode = This.HitTest(X * Thisform.nTreeFactorX, Y * Thisform.nTreeFactorX)
    * If you have a valid node, select it.
    if not isnull(loNode)
        loNode.Selected = .T.
        This.NodeClick(loNode)
    endif not isnull(loNode)
    * show the right-click menu now
else
    * handle any left-mouse click stuff necessary
endif Button = 2
```
Add: adds a new node and returns a reference to it. It uses the following syntax:

```vbnet
<Object>.Nodes.Add(Relative, Relationship, Key, Text, Image, SelectedImage)
```

-- Relative is the index or key of an existing node object. If it isn't specified, the new node is placed at the end of the top node hierarchy.
-- Relationship indicates where the new node should be placed relative to the node specified in the first parameter:

1. The node is placed after all other nodes at the same level of the relative node.
2. The node is placed after the relative node.
3. The node is placed before the relative node.
4. The node becomes a child of the relative node.

-- Key is a unique string used to identify the node. If the tree is loaded with records from a table, I use the primary key of the record (converted to a string if necessary). Otherwise, I just use a sequential number converted to a string.
-- Text is the text displayed in the control for the node.
-- Image is the index number of an image in the associated ImageList control.
-- SelectedImage is the index number of an image in the associated ImageList control that's shown when the node is selected.

Clear: nukes all nodes.

Remove: removes the node corresponding to the specified index.

Nodes collection properties
Here are the properties of the Nodes collection:

- Count: the number of nodes.
- [<Index>]: an object reference to node number <Index>.

Node objects methods
Here are the methods of node objects:

- CreateDragImage: can't be used in VFP because DragIcon expects a CUR file name while this method returns an image.
- EnsureVisible: ensures that the specified node is visible. This method scrolls the TreeView control if necessary and expands all parent nodes of the specified node.

Node objects' properties
Here are the properties of node objects:

- Children: .T. if the node object has any child node objects.
- Expanded: .T. if the node object is expanded.
- FullPath: the concatenation of the Text values of all parent objects leading to this one, separated by the control's PathSeparator property. It's very similar to the concept of a fully qualified file path and name.
- Image, ExpandedImage, and SelectedImage: the appropriate image number in the associated ImageList control.
- Index: the index of the node object in the Nodes collection.
- Key: the unique key assigned when the node was added.
- Child, FirstSibling, LastSibling, Previous, Parent, Next, and Root: point to the appropriate node object relative to the specified one.
- Selected: .T. if the node object is selected. Setting this property to .T. automatically highlights the node and sets the Selected property of the previously selected node to .F.
- Text: the text that appears in the control.

Loading the TreeView control
While it's relatively simple to load the tree, it can have some interesting twists. Here's a straightforward example. Assume that you want to load a tree with customers and orders for those customers (as in the TREEVIEW sample form). The following code will do this; the code would likely get called during form initialization (for example, in a custom method of the form called in the Init() method of the form or the Init() method of the TreeView control):

```vbnet
with This
  *
  Set up the CUSTOMER and ORDERS tables.

  select CUSTOMER
  set order to CUST_ID in ORDERS
  set order to COMPANY
  scan
```

http://foxtalknewsletter.com/ME2/Audiences/Segments/Publications/Print.asp?Module=... 11.02.06
The key assigned to each node is the key for the record plus a prefix to indicate what table the record comes from ("C" for CUSTOMER and "O" for ORDER). The prefix avoids the problems where an order and customer have the same primary key. The combination of prefix and primary key allows you to quickly find the record matching a particular node (using the node's Key property, it gets the table from the first character, and the primary key to seek from the rest of the string).

The only downside to this code is that it must process every order for every customer before the form can be displayed. If there are a lot of customers or orders, this can take a long time. A better approach is to load only the top-level items (customers in this case), then in the Expand() method, check whether the node about to be expanded has been loaded with child nodes. If it hasn't, load the child nodes just for the current node. This is much faster than loading the entire tree at the beginning. One tip: if you don't load any children for a node, no plus sign (+) will appear for the node. You'll need to add at least one node, even a "dummy" one, for each top-level node. When the node is expanded for the first time, remove the dummy node and then add the "real" child nodes.

The TREEVIEW sample form shows how to use this technique. It has an ILoadChildren property that determines whether all orders and products are loaded before the form is displayed. Try setting this to .F. and notice how long it takes the form to display.

You might need to reload the tree at some point. For example, if other users are adding customers or orders, you might need to refresh the tree from time to time with any new customers or orders. To prevent the hassles of figuring out which records are already in the tree and which aren't, you could use the Nodes collection's Clear() method to clear the tree, then reload it from scratch. The only problem with this approach is that some nodes may have been expanded. Clearing and reloading the tree causes all nodes to start in the default collapsed view. To overcome this, you can save the Expanded property, and either the Key or FullPath properties, of each existing node in an array or cursor, and clear and reload the tree. Then go through the array or cursor and reset the Expanded property of those previous nodes to the saved former value. The GetExpanded and SetExpanded methods of the TREEVIEW sample form do just that.

Drag and drop

Based on messages I've seen on CompuServe, there's a lot of confusion about whether you can drag and drop with the TreeView control. Some folks think that because VFP doesn't implement drag and drop with other applications such as Word (which is true), it can't do it with ActiveX controls (which isn't true). You can drag and drop with ActiveX controls even though they're non-VFP applications because when you drop an ActiveX control on a form, VFP places it inside a VFP container, which, of course, does support drag and drop. As a result, dragging and dropping with a TreeView control (either from or to the control) works no differently than it does for any other control.

The TREEVIEW sample form implements drag and drop, both from and to the control. You can drag a node from the tree to the text box under the "Dragged Node" label; the text of the node will appear in the text box. You can also drag the "Drag me to tree" label and drop it on any node in the tree; a new node will be added, showing the text of the node you dropped the label on, along with the date and time.

Conclusion

The TreeView ActiveX control is a wonderful addition to your VFP "toolbox." It allows you to implement a visual control for any hierarchical structure -- a control your users probably already know how to use. Once you get past some of the quirks and poor documentation of the TreeView control, you'll find a ton of uses for it. I hope this article sets you on the path to successfully implementing this control.