Custom UI Controls: SFComboTree

Doug Hennig

Sometimes you need to display a list of items but don't have much room to do so. While a combo box is usually used in this case, it doesn't display a hierarchical list or support checkboxes for items. SFComboTree fits this need nicely.

Last issue, I started a series on custom UI controls I frequently use, beginning with a splitter control. This time, let's look at the SFComboTree control.

SFComboTree

SFComboTree is so named because it combines a VFP ComboBox control with a Microsoft TreeView ActiveX control. Although it can be used for a variety of needs, SFComboTree is most useful for two specific tasks: a hierarchical list of data and multiple checkboxes. In both cases, its compact size makes it ideal for forms lacking space for a large control like a list box or TreeView.

In its "closed" state, SFComboTree looks like a combo box, so it's only 24 pixels high and only as wide as you size it. When you "open" the control, it's as tall as you wish, temporarily overlapping any other controls as necessary. **Figure 1** shows a sample form with two closed SFComboTree controls. In **Figure 2**, the top control is open so it displays a hierarchical list of folders and temporarily covers the control below it. **Figure 3** shows an example of a list of multiple checkboxes.

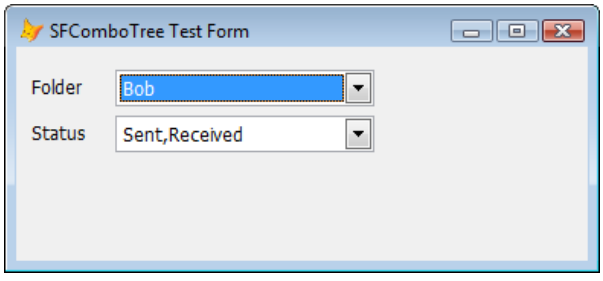

Figure 1. When it's closed, SFComboTree takes up very little space.

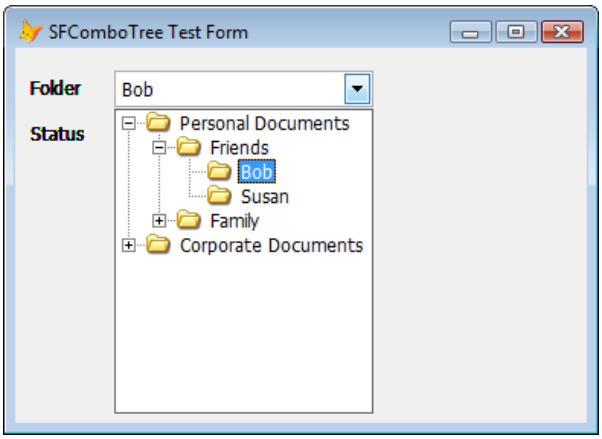

Figure 2. SFComboTree provides a combined combo box and TreeView control so the user can easily view a hierarchical list.

Here are some of the characteristics of SFComboTree:

- The combo box displays the text of the selected node in the TreeView. However, as you'll see later, you can override that to, for example, display a comma-delimited list of all checked items.
- Programmatically, you can read from or write to the Value property to find out which node is selected or to automatically select a node (such as displaying a value from an existing record).
- Nodes in the TreeView control can have images or not, checkboxes or not, and be arranged hierarchically or not.
- The control automatically resizes when its form or container does. Although the combo box doesn't get taller, it does get wider. The TreeView resizes both vertically and horizontally, regardless of whether the control is opened or closed when the form is resized.
- Clicking the combo box's down arrow opens the control. You can configure how the control closes: clicking the down arrow again, when the user clicks a node, when the user

double-clicks a node, or when the control loses focus.

Using SFComboTree

To use SFComboTree, drop it on a form and set the properties shown in **Table 1** as desired. Fill in the LoadTree and LoadImages methods with code that loads the nodes in the TreeView and images used by the TreeView, respectively. Leave LoadImages empty if you don't want images in the TreeView (for example, if you're using it with checkboxes).

Table 1. Important properties of SFComboTree.

Property	Description
FontName	The font to use for the combo box and TreeView (default Tahoma).
FontSize	The font size (default 9).
ToolTipText	The tool tip text for the container and combo box.
lCloseOnClick	.T. to close the control when the user clicks an item in the TreeView (default .F.).
lCloseOnDblClick	.T. to close the control when the user double-clicks an item in the TreeView (default .F.).
lLoadImagesOnInit	.T. to load images when the control is initialized (default .F.).
lLoadTreeOnInit	.T. to load the TreeView when the control is initialized (default .F.).
lMoveToBack	.T. to set the ZOrder of the control to the back when it's closed (default .T.)
lNoClose	.T. to not have LostFocus close the control (default .F.).

If you need to do some setup tasks before loading images into the ImageList (for example, you have to wait until the form the control is on has initialized), set lLoadImagesOnInit to .F. and call LoadImages manually once the setup is done. Otherwise, set that property to .T. so LoadImages is called from Init.

Init calls LoadTree to load the TreeView if lLoadTreeOnInit is .T. That isn't strictly necessary because opening the control calls LoadTree if the TreeView doesn't have any nodes. However, if for some reason you want the TreeView loaded earlier, set lLoadTreeOnInit to .T.

TreeView controls are notoriously slow for loading if you have a lot of nodes. To improve performance when loading hierarchical nodes into the TreeView, you can just load the top-level nodes in LoadTree. SFComboTree can then load the child nodes for a top-level node when it's expanded for the first time. Of course, you'll need to load at least one child node for every top-level node or the "+" won't appear for the node. To make this work, create a child node under each top-level node with "Loading…" as the text for the node. When the user expands a node, the TreeView's Expand event fires, and code in that event calls SFComboTree's LoadExpandedNode method if it finds a "Loading…" child node. Fill in the code in LoadExpandedNode to load the child nodes for the specified parent node. The net result is that the child nodes are only loaded the first time a parent node is expanded, which is much less of a performance hit than loading all nodes at one time, regardless of whether the user will ever expand the parent nodes or not.

You can change any of the properties of oTree or oImageList as necessary. For example, if you want checkboxes, set oTree.CheckBoxes to .T.

If you have code you want to execute when the user selects an item, put the code into the ItemSelected method. Otherwise, you can check the lChanged (.T. if the user changed the selected node in the TreeView) and Value (the text of the selected node) properties as necessary, such as when the user saves a record or closes the form.

Try it out

The sample TestComboTree.SCX included with the downloads for this article shows a couple of typical uses for SFComboTree: a hierarchical list and multiple checkboxes. The top instance, named oFolder, displays a list of "folders" as stored in a table named FOLDERS.DBF, some of which are children of other folders. oFolder has lCloseOnClick set to .T. so the control closes when the user selects a node.

LoadTree has the following code:

```
local lcKey, ;
  lcName, ;
  lcParentKey

* Open the Folders table if necessary.

if used('FOLDERS')
  select FOLDERS
else
  select 0
  use FOLDERS again shared
endif used('FOLDERS')

* Go through each record and create a node in
* the TreeView under the appropriate parent
* node.

with This.oTree
  scan
    lcKey       = 'F' + transform(ID)
    lcName      = trim(NAME)
    lcParentKey = 'F' + transform(PARENT)
    if empty(PARENT)
      loNode = .Nodes.Add(, 1, lcKey, ;
        lcName, 'Folder')
    else
      loNode = .Nodes.Add(lcParentKey, 4, ;
        lcKey, lcName, 'Folder')
```

```
    endif empty(PARENT)
  endscan
endwith
```

LoadImages just loads a single image into the ImageList control. Because lLoadImagesOnInit is .T., oFolder loads the image from Init.

```
This.oImageList.ListImages.Add(1, 'Folder', ;
  loadpicture('Folder.bmp'))
```

ItemSelected simply displays the folder the user selected by passing This.Value to MESSAGEBOX().

Figure 2 shows the form when oFolder is open. Because the TreeView is loaded from the Folders table, it serves as an example for a dynamically loaded hierarchical list.

The second SFComboTree on the form, named oStatus, shows how to create a list of items with checkboxes. Its LoadTree method loads a hard-coded list of status values (which could easily be loaded from a table in a real application) but also turns on checkboxes for nodes. This allows you to select multiple status values, such as "Sent" and "Received".

```
local loNode
with This.oTree
  .Checkboxes = .T.
  loNode = .Nodes.Add(, 1, 'S1', 'Sent')
  loNode = .Nodes.Add(, 1, 'S2', 'Filed')
  loNode = .Nodes.Add(, 1, 'S3', 'Received')
endwith
```

Because we want the combo box to show all selected items, not just the last one clicked, ItemSelected concatenates all checked items into a comma-delimited list and sets Value to that list.

```
local lcStatus, ;
  lnI, ;
  loNode
with This.oTree
  lcStatus = ''
  for lnI = 1 to .Nodes.Count
    loNode = .Nodes(lnI)
    if loNode.Checked
      lcStatus = lcStatus + ;
        iif(empty(lcStatus), '', ',') + ;
        loNode.Text
    endif loNode.Checked
  next lnI
endwith
This.Value = lcStatus
```

Figure 3 shows the form when you open oStatus.

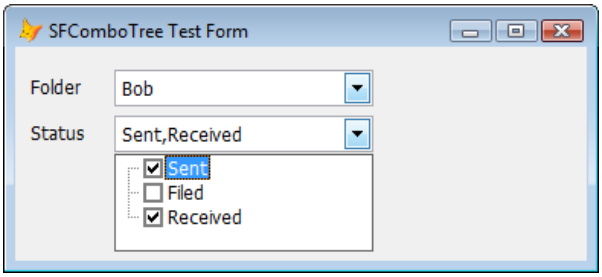

Figure 3. The second SFComboTree in the sample form demonstrates a list of items with checkboxes.

SFComboTree details

SFComboTree, defined in SFComboTree.VCX, is subclass of Container with four controls: a ComboBox named cboCombo, a TreeView control named oTree, an ImageList control named oImageList, and a shape named shpTreeView that provides a border for the TreeView control. The container has BorderWidth set to 0 so it doesn't appear as a container and Height set to 24, the same height as cboCombo.

cboCombo has RowSourceType set to 1-Value and RowSource set to nothing because we don't need the combo box to contain a list of choices; it simply provides the visual appearance of a text box and a drop down arrow. Style is 2-Dropdown List so the user can't type a value.

The custom changes to oTree's properties control its appearance and behavior. Appearance is 0-Flat, HideSelection is .F., HotTracking is .T., Indentation is 10, LabelEdit is 1-Manual (so the user can't edit the nodes), and LineStyle is 1-RootLines.

The Init method of SFComboTree sets up the controls

```
with This

* Set up the combobox.

  .cboCombo.Anchor      = 0
  .cboCombo.Width       = .Width
  .cboCombo.Anchor      = .Anchor
  .cboCombo.ToolTipText = .ToolTipText

* Save the current height of the control and
* the form and our Anchor value.

  dimension .aParentHeights[1, 2]
  .aParentHeights[1, 1] = .Height
  .aParentHeights[1, 2] = .Height
  .nInitialFormHeight   = Thisform.Height
  .nSavedAnchor         = .Anchor

* Set our font name and size to their own
* values so Assign takes care of setting the
* other controls.

  .FontName = .FontName
  .FontSize = .FontSize

* Call CloseControl so everything is sized
* properly for a closed appearance.

  .CloseControl()
```

```
* If we're supposed to, load the images now.

  if .lLoadImagesOnInit
    .LoadImages()
  endif .lLoadImagesOnInit

* If we have any images, use the ImageList
* control with the TreeView.

  if .oImageList.ListImages.Count > 0
    .oTree.Object.ImageList = .oImageList
  endif .oImageList.ListImages.Count > 0

* If we're supposed to, load the TreeView now.

  if .lLoadTreeOnInit
    .LoadTree()
  endif .lLoadTreeOnInit
endwith
```

The DropDown event of cboCombo, fired when the user clicks the down arrow, opens or closes the control by calling SFComboTree's OpenControl or CloseControl methods. There's also some code in that event to handle what may be a bug in VFP; see the comments for details.

OpenControl is a fairly complex but well-documented method. It's responsible for adjusting the control so the TreeView is visible and sized appropriately, and ensuring the selected node in the TreeView matches the value displayed in the combo box.

```
local lnAnchor, ;
  loParent, ;
  lnParent, ;
  lnI, ;
  loNode
with This

* If we haven't already done so, load the
* TreeView the first time we're opened.

  if .oTree.Nodes.Count = 0
    .LoadTree()
  endif .oTree.Nodes.Count = 0

* If we loaded images later than from Init,
* use the ImageList control with the
* TreeView.

  if not .lLoadImagesOnInit and ;
    .oImageList.ListImages.Count > 0
    .oTree.Object.ImageList = .oImageList
  endif not .lLoadImagesOnInit ...

* Turn off anchoring since we'll be resizing
* and moving controls.

  lnAnchor = .Anchor
  store 0 to .Anchor, .cboCombo.Anchor, ;
    .oTree.Anchor, .shpTreeView.Anchor

* Save our height, then set it to the desired
* height, accounting for any resize of the
* form.

  .aParentHeights[1, 1] = .Height
  .Height               = ;
    min(.nOriginalHeight + Thisform.Height - ;
    .nInitialFormHeight, Thisform.Height - ;
    This.Top - 5)
  .aParentHeights[1, 2] = .Height

* Save the height of all parent containers and
* adjust them if necessary. Also, save the
* current anchor values and add 5 if necessary
* so they resize vertically.

  loParent = This.Parent
  lnParent = 1
  do while vartype(loParent) = 'O' and ;
    lower(loParent.BaseClass) = 'container'
    lnParent = lnParent + 1
    dimension .aParentHeights[lnParent, 3]
    if loParent.Height < .Top + .Height
      .aParentHeights[lnParent, 3] = ;
        loParent.Anchor
      loParent.Anchor              = 0
      .aParentHeights[lnParent, 1] = ;
        loParent.Height
      loParent.Height = .Top + .Height
      .aParentHeights[lnParent, 2] = ;
        loParent.Height
      loParent.Anchor = ;
        .aParentHeights[lnParent, 3]
      if not bittest(loParent.Anchor, 0) and ;
        not bittest(loParent.Anchor, 2)
        loParent.Anchor = loParent.Anchor + 5
      endif not bittest(loParent.Anchor ...
    endif loParent.Height < .Top + .Height
    loParent = loParent.Parent
  enddo while vartype(loParent) = 'O' ...

* Adjust the size of the TreeView and shape in
* case the container was resized while we were
* closed.

  .oTree.Width        = .Width - 2
  .oTree.Height       = .Height - ;
    .cboCombo.Height - 4
  .oTree.Left         = .shpTreeView.Left + 1
  .oTree.Top          = .shpTreeView.Top + 1
  .shpTreeView.Width  = .Width
  .shpTreeView.Height = .Height - ;
    .cboCombo.Height - 2

* If the current value doesn't match the
* selected item in the TreeView, find and
* select the appropriate item.

  if vartype(.oTree.SelectedItem) <> 'O' or ;
    (not empty(.cboCombo.DisplayValue) and ;
    not .cboCombo.DisplayValue == ;
      .oTree.SelectedItem.Text)
    for lnI = 1 to .oTree.Nodes.Count
      loNode = .oTree.Nodes.Item(lnI)
      if .cboCombo.DisplayValue == loNode.Text
        loNode.Selected = .T.
        exit
      endif .cboCombo.DisplayValue ...
    next lnI
  endif vartype(.oTree.SelectedItem) ...

* Enable the controls appropriately, then set
* focus to the TreeView.

  .oTree.Visible       = .T.
  .shpTreeView.Visible = .T.
  .ZOrder(0)
  .shpTreeView.ZOrder(0)
  .oTree.ZOrder(0)
  .lComboTreeOpen = .T.
  .oTree.SetFocus()

* Restore anchoring and add 5 to it so we
* resize vertically.

  .cboCombo.Anchor = lnAnchor
  store lnAnchor + 5 to .Anchor, ;
    .oTree.Anchor, .shpTreeView.Anchor
endwith
```

I won't show the code for CloseControl for space reasons; it too is well-documented and should be easy enough to understand. It has to do the opposite of OpenControl: reset the height of the control so the TreeView is no longer visible. CloseControl also calls the abstract ItemSelected method so you can add some code in a subclass or instance to do something when the user closes the control.

The custom FontName and FontSize controls have Assign methods so changing the font for the control changes it for the combo box and TreeView. Enabled also has an Assign method for similar reasons. The custom Value property has Access and Assign methods that read from and write to cboCombo.DisplayValue so you can simply reference Control.Value rather than Control.cboCombo.DisplayValue.

Summary

I use SFComboTree in lots of places in my applications. It's even used to display the hierarchy of controls for a form or class in PEM Editor, a very cool replacement for the VFP New Property, New Method, and Edit Property/Method dialogs, available from VFPX (http://vfpx.codeplex.com). All of those uses have one thing in common: the need to display a list of items (hierarchical or not) while taking up very little space in a form.

Doug Hennig is a partner with Stonefield Systems Group Inc. and Stonefield Software Inc. He is the author of the award-winning Stonefield Database Toolkit (SDT); the award-winning Stonefield Query; the MemberData Editor, Anchor Editor, and CursorAdapter and DataEnvironment builders that come with Microsoft Visual FoxPro; and the My namespace and updated Upsizing Wizard in Sedna.

Doug is co-author of "Making Sense of Sedna and SP2," the "What's New in Visual FoxPro" series (the latest being "What's New in Nine"), "Visual FoxPro Best Practices For The Next Ten Years," and "The Hacker's Guide to Visual FoxPro 7.0." He was the technical editor of "The Hacker's Guide to Visual FoxPro 6.0" and "The Fundamentals." All of these books are from Hentzenwerke Publishing (http://www.hentzenwerke.com). He wrote over 100 articles in 10 years for FoxTalk and has written numerous articles in FoxPro Advisor, Advisor Guide to Visual FoxPro, and CoDe. He currently writes for FoxRockX (http://www.foxrockx.com).

Doug spoke at every Microsoft FoxPro Developers Conference (DevCon) starting in 1997 and at user groups and developer conferences all over the world. He is one of the organizers of the annual Southwest Fox conference (http://www.swfox.net). He is one of the administrators for the VFPX VFP community extensions Web site (http://vfpx.codeplex.com). He has been a Microsoft Most Valuable Professional (MVP) since 1996. Doug was awarded the 2006 FoxPro Community Lifetime Achievement Award (http://fox.wikis.com/wc.dll?Wiki~FoxProCommunity LifetimeAchievementAward~VFP).