Try Thor's Terrific Tools, Part 2

Thor offers lots of tools for working with classes and forms. Learning to use them can make you more productive.

Tamar E. Granor, Ph.D.

In my last article, I showed a few of the tools that are included with Thor. This time, I'll look at some more of them, including a number that simplify refactoring.

Thor is a VFPX tool whose purpose is to make it easier to use other tools. It provides an easy way to put your homegrown tools on the VFP menu, as well as to make them available via hotkeys. It grew out of the VFPX PEM Editor project.

When you install Thor, it includes dozens of tools for tasks from grabbing the list of fields in a table to adding local declarations to code. (The Create Locals tool is discussed in my last article.) Even if you have no homegrown tools to add, these built-in tools make it worth installing and running Thor.

Most of the tools operate on code or other items in the VFP IDE, so I'll need to demonstrate on some programs, forms and classes. As much as possible, I'll use code that comes with VFP.

For each tool, I'll also show you where it's located on the Thor Tools menu.

Copying and pasting PEMs

While you can copy an object in VFP and paste it onto another form or class, there's no native way to give one object the same property values and method code as another object. A pair of Thor tools provides that capability.

Have you ever needed to configure a control to be the same or almost the same as an existing control (in a case where using a common class doesn't make sense)? Configuring the new control to match the existing one is something of a pain, as you have to go one by one through the changed PEMs (properties, events and methods) in the Property Sheet for the original object, and for each, switch to the new object to set its corresponding PEM.

Thor takes the pain away. It also makes it easy to insert a parent class into an inheritance hierarchy.

Copy (for comparing and pasting)

Menu: Objects and PEMs | Copy / Paste | Copy (for comparing and pasting)

This tool lets you pick up all the modified PEMs for an object and store them on a special clipboard. To use it, you simply select the object and choose this tool.

By itself, this tool isn't terribly useful. It's meant to be followed by Paste properties and method code (described in the next section) or Compare with copied object (described later in this article).

Paste properties and method code

Menu: Objects and PEMs | Copy / Paste | Paste properties and method code

This tool lets you set properties and methods to the values you previously copied from another object. You can choose which PEMs to copy and whether to add properties and methods that don't exist in the target object. Note that the target object does not have to be based on the same base class as the copied object.

Figure 1 shows a form that lets the user select a product (from the Northwind Products table) using a combo. The combo is based on the VFP base combobox class. Suppose once I've created this form, I decide that I want to insert a combo class in between the base class and the one on the form. If I want the class to be identical to the combo on the

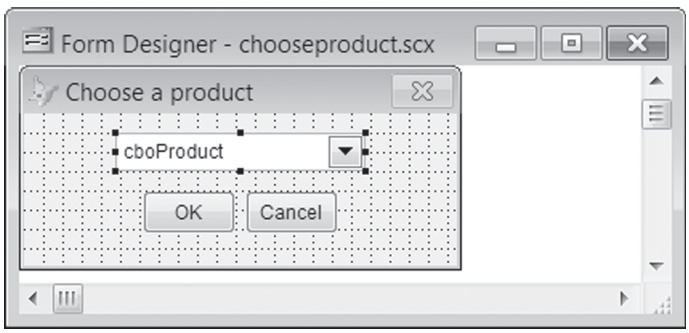

Figure 1. This form uses a combobox to let the user choose a product.

form, I can just select the combo and choose File | Save As Class from the menu. But if I want to do something more nuanced, this tool is just the ticket.

After selecting the combo, choose the Copy (for comparing and pasting) tool to pick up its properties and method code. Then, create the new class (using your favorite technique for doing so). Next, use the Paste properties and method code tool; the form in Figure 2 appears.

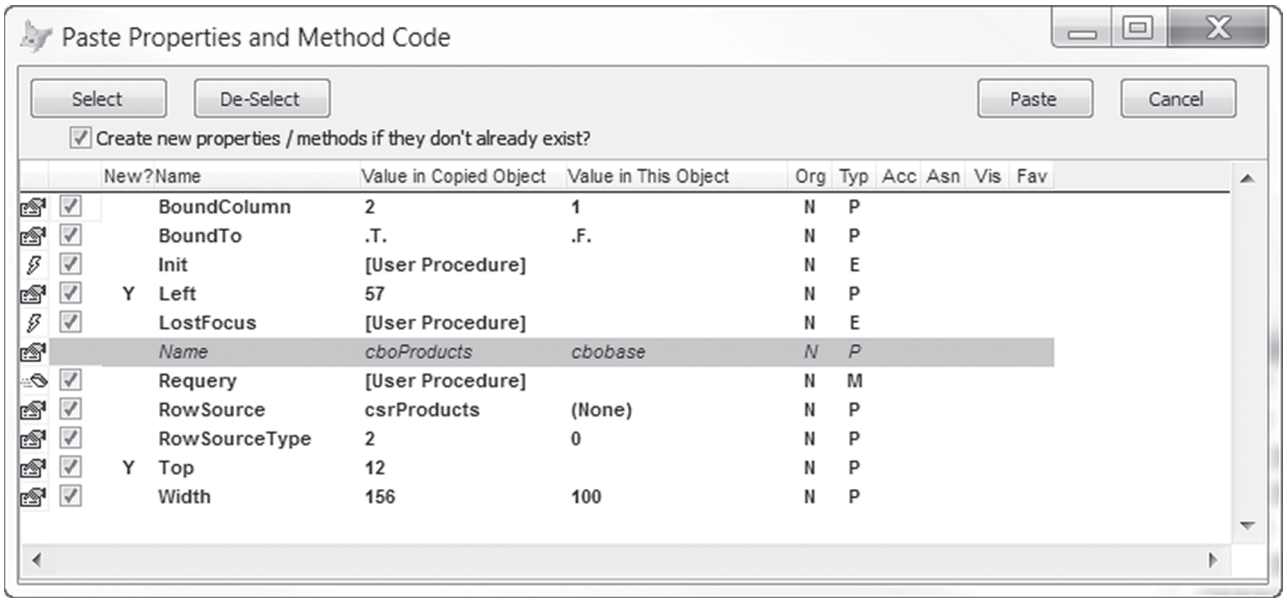

Figure 2. The Paste properties and method code tool displays this form, which allows you to choose which PEMs are copied to the target object.

You can choose which PEMs to copy to the target. In the example, you probably don't want to copy the Left and Top properties, since the class shouldn't come with a specified position Once you've chosen the ones you want, click the Paste button.

When the target is not a form or a class (that is, the class that's being edited in the Class Designer), properties that exist in the source, but not in the target are marked as New and can't be copied. However, when the target is a form or a class as in the example, a checkbox asks whether to create PEMs that don't exist.

Comparing objects

One of my favorite non-VFP tools is Beyond Compare, which lets me compare folders and files to find differences between them. With Frank Perez's VFP add-on (http://pfsolutions-mi.com/Product/VFP2Text), I can even compare VFP classes and forms. But this isn't a handy way to do so when I'm in the middle of editing them, since you have to close the files to look at them with Beyond Compare.

Thor includes two different tools for comparing objects; both are pretty useful. Compare with Parent Class lets you see which properties of a class have non-default values and shows you the parent class's values for the same properties. Compare with copied object lets you compare two unrelated objects.

Compare with Parent Class

Menu: Parent Classes | Compare with Parent Class

Although the VFP Property Sheet lets you see which PEMs of an object have been changed from their default values (and even lets you see only non-default PEMs), it provides no easy way to see what those values are in the parent class.

The Compare with Parent Class tool opens a separate form that shows each non-default property, event and method in the selected object, along with its value in the selected object and in the parent class. It also indicates those properties with the same value in both places. Finally, it allows you to reset any of the displayed PEMs of the selected object to their default values.

Figure 3 shows the form DataNav.SCX from the Solution Samples that come with VFP. The _tablenav object is selected. Figure 4 shows the form opened by the Compare with Parent Class tool (which is clearly a variation of the one used for the Paste properties and method code tool). It indicates that three properties and five methods have been set in the form's Property Sheet. One of those, Name, is the same on the form as in the class.

You can use the checkboxes to select some or all of the PEMs shown and then click Reset Selected Items to Default to clear those PEMs in the selected object. Properties that have the same value as in the class are automatically checked.

The Select and De-Select buttons open a dropdown menu (shown in Figure 5) that lets you specify the type of item to check or uncheck.

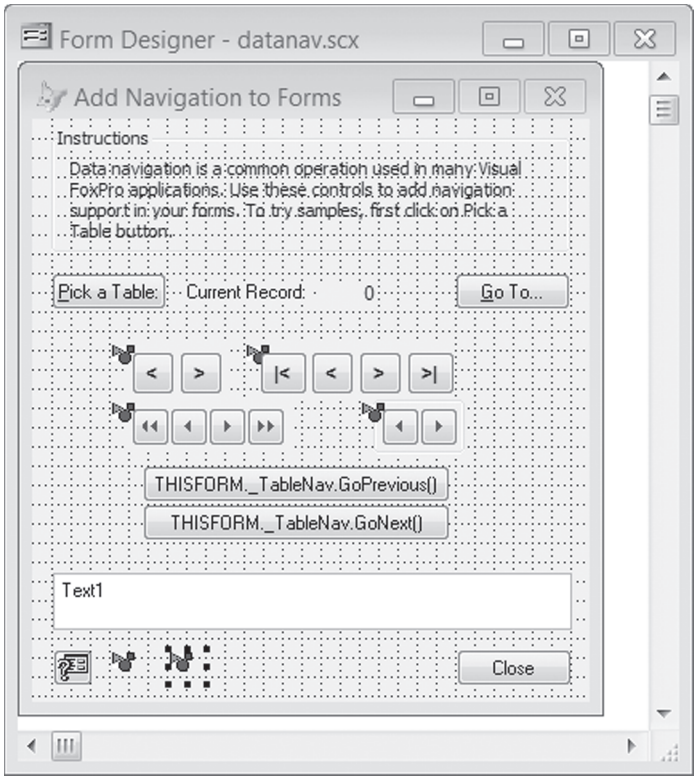

Figure 3. In this form from the Solutions Samples, an object based on the _tablenav class (from the FoxPro Foundation Classes) is selected.

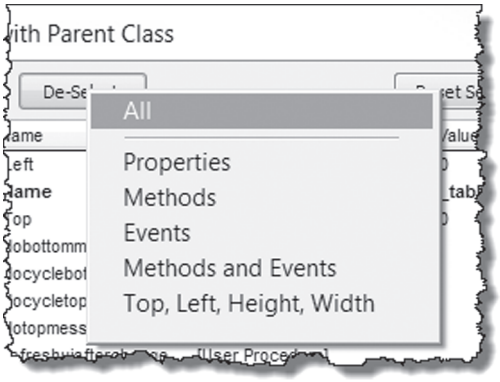

Figure 5. The Select and De-Select buttons of the Compare with Parent Class tool let you choose the type of item to check or uncheck.

Compare with copied object

Menu: Objects and PEMS | Copy / Paste | Compare with copied object

The second Thor tool for code comparison is part of the set of copy-and-paste tools for classes described in "Copying and pasting PEMs," earlier in this article.

To use this tool, you must first select an object and use the Copy (for comparing and pasting) tool found on the same branch of the Thor menu. Then select the object to which you want to compare the first object and choose this tool.

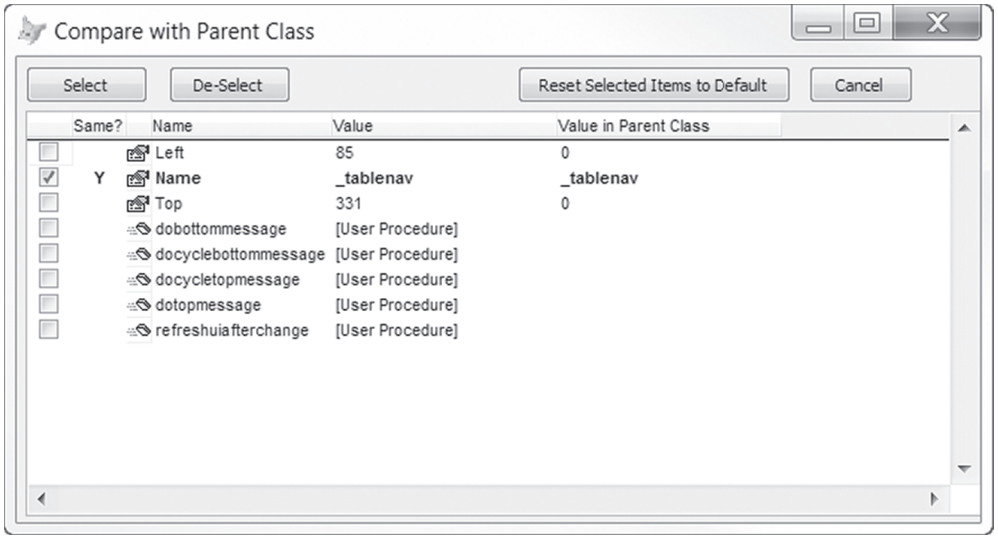

Figure 4. The Compare with Parent Class tool shows only those properties whose values are non-default.

To demonstrate, we'll look at the same object we used for Compare with Parent Class, even though that tool is a better choice in this case. Starting with the DataNav form, click on the _tablenav object and choose Thor Tools | Objects and PEMs | Copy / Paste | Copy (for comparing and pasting). Now close the form and open the _tablenav class from _table.vcx in the FFC folder (or better yet, use the Edit Parent and Containing Classes tool described in my last article to open it). When you choose this tool from the menu, the form shown in Figure 6 appears.

There's a reason this tool points out properties that have the same value as in their parent class. Every property that's set in the Property Sheet has to be evaluated and assigned during initialization of a form or other class. So having properties that are explicitly set to the default value from the parent class can slow execution down (though you're unlikely to notice it unless there are many objects with many such properties).

PEMs with a gray background don't exist in the object where the value column is also gray. In the example, the _tablenav class doesn't have Left and Top properties, but when dropped on the form,

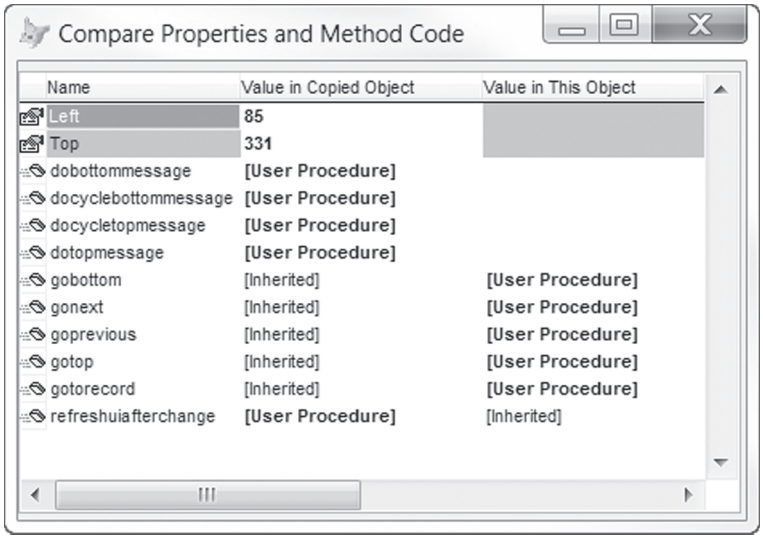

Figure 6. This form appears when you use the Compare with copied object tool. It shows PEMs that are different in the two objects.

they're added. Methods where a column is blank (like the dobottommessage method in the example) exist, but have no code anywhere in the inheritance hierarchy up to that class.

This tool seems particularly useful for situations where one instance of a class isn't working, but other instances are. You can compare the PEMs of a working instance to those of the non-working one to see what you've done differently.

It also seems useful for figuring out whether two existing classes might productively share a common parent class. (When they can, the Paste properties and method code tool gives you an easy way to jump-start creation of the parent class.)

Re-Define Parent Class

Menu: Parent Class | Re-Define Parent Class

Changing the class a control is based on has always been a bit of a pain. The Class Browser lets you change a form's or class's parent class, though not the parent for a control already on another form or class. Various third-party tools like HackCX provide a way to change the parent of any object. But all of these approaches require you to close the class or form you're working on so that the tool can open it. This tool lets you change the parent class of a control on a form or another class without closing the form or class.

To use it, select the object whose parent class you want to change, and choose this tool from the menu. The form shown in Figure 7 appears; you use it to find the new parent class. The form lets you specify the type of class you're looking for (indicated by the arrow) and where to look for it. The type can be a specific base class, or "<All>." For scope, you can specify folder (as in the figure) or choose any of the projects on the MRU list. You can also specify all or part of the name of the class you want; unlike the usual VFP string comparison, the portion you specify can be anywhere in the class name. Similarly, you can provide part of a file name to limit the search.

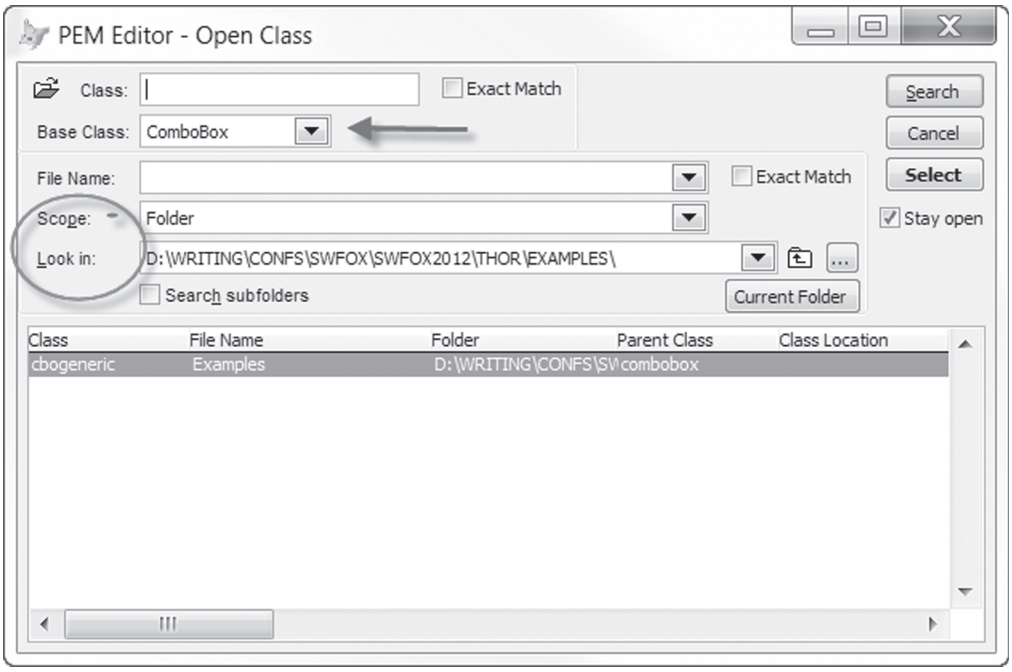

Figure 7. Find and select the new parent class for an object using this form.

When you've set up the type of class and where to look, click the Search button and the grid at the bottom of the form shows all classes that match your specifications. Choose the one you want and click the Select button to change the parent of the selected object to that class.

The same dialog used for the Paste properties and method code and the Compare with parent class tools opens, as in Figure 8, showing you properties and methods that are different in the new parent and the existing object. Decide which of the changed values you want to keep and click Paste.

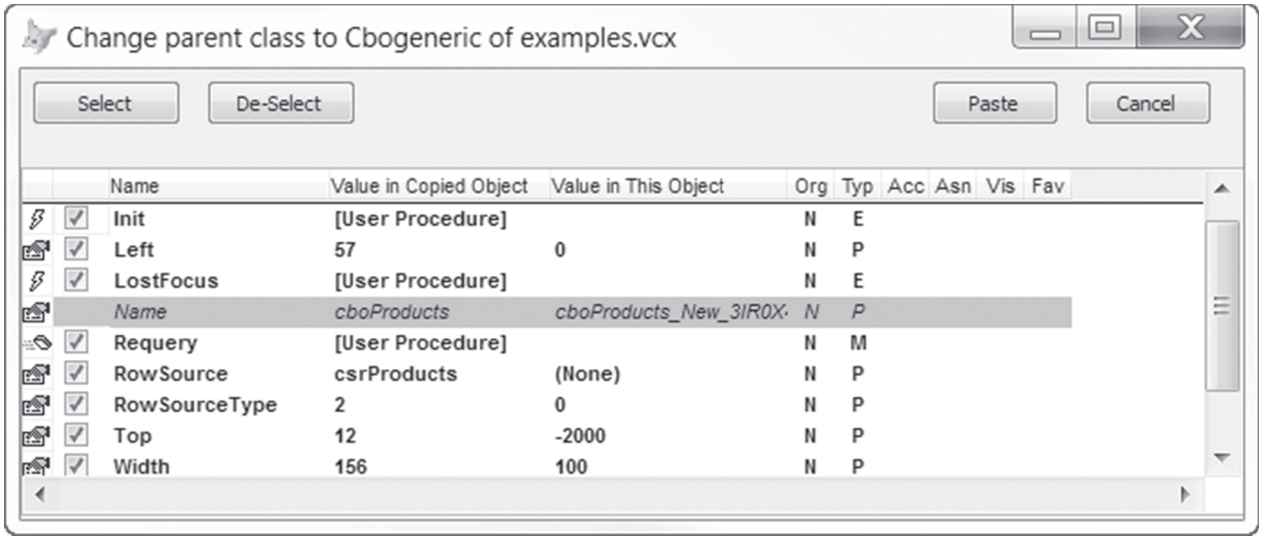

Figure 8. Once you choose the new parent class, you have the opportunity to decide which property values and method code should be kept.

Document Treeview

Menu: Applications | Document Treeview

When Document View was added in VFP 7, it gave us an easy way to navigate inside files containing multiple routines. It was a major improvement over the Procedures and Functions List it replaced. I use it all the time when working with classes created in code (PRG).

But Document View has never been all that helpful for forms or for classes stored in a class library (VCX). While it lists every method that contains code, it doesn't give any sense of structure, and in a busy form or class, it can be quite cluttered.

The Document Treeview tool is better suited for visual classes (and, in fact, doesn't work for code). Document Treeview is based on the main combobox of PEM Editor, but you can use it as a stand-alone tool. It shows the objects in a form or class and those of their methods that contain code. As the name indicates, it uses a treeview control to organize the information, so you can expand or collapse any section. It can be resized as well as docked. In addition, you can control what appears at any time.

Figure 9 shows Document Treeview for the form ToolboxAddClassLib that's part of the Toolbox (found in folder tools\xsource\vfpsource\toolbox\of your VFP installation after you've unzipped XSource.ZIP).

The colors in Document Treeview help to quickly understand the display. Names of objects are shown in black. If the object was inherited along with its container from a parent class (like the controls inside oClassLib in the example), its backcolor is gray. Method names are shown in blue if they contain code at this level; if they have only inherited code, they're shown in gray. All method names are bold, which offers another way to distinguish them from object names.

Clicking on an object makes it the current object in PEM Editor, if that tool is open. Often, it also selects that object in the Form or Class Designer and makes it the current object in the Property Sheet; there are cases where it's not possible to do so programmatically, however.

You can click on any method to open it for editing; that's just like Document View. But Document Treeview offers another possibility. Use Ctrl+Click on the method and a window opens showing you all code for that method at all levels of the inheri-

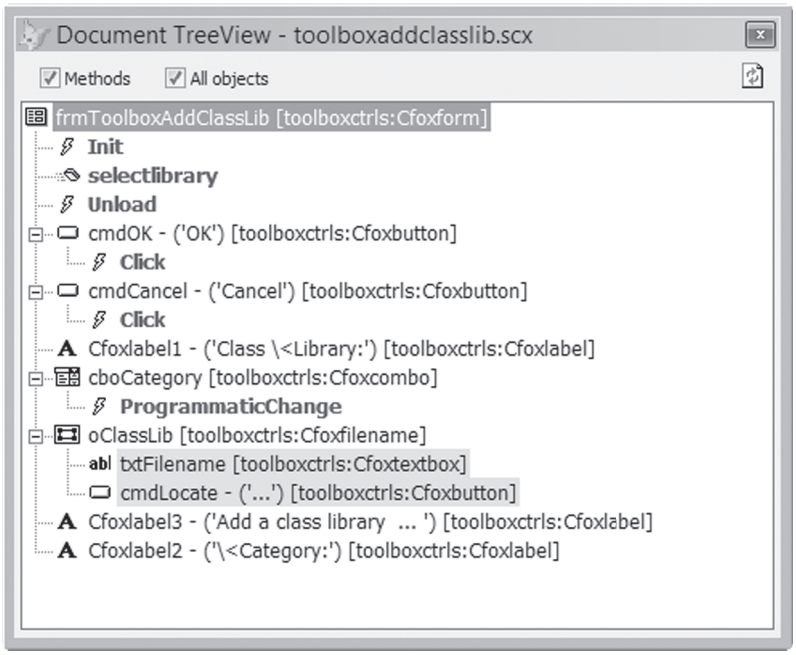

Figure 9. Document Treeview shows you the objects in a class or form and any methods that contain code.

tance hierarchy. Of course, you can't edit the code there, but it's a much easier way to see all the code than anything offered natively. (Another Thor Tool, Code Listings, also lets you see all the code in one place.)

Figure 10 shows part of the window that opens when you Ctrl+Click on the Init method of the ToolboxAddClassLib form. The code added at the form level is shown first, followed by the code inherited from the cFoxForm class. If there were more classes in the inheritance hierarchy, they'd be included, as well. For custom methods, the hierarchy is followed back to the class to which the method was added.

There are a number of ways to control what appears in Document Treeview. The two checkboxes above the treeview control offer three variations. When both are checked, you see all objects, as well as those methods that have code, as in Figure 9. If you uncheck All objects, you see only those objects that have any code, as well as the methods that contain code. Figure 11 shows Document Treeview for the ToolboxAddClassLib form with All objects unchecked.

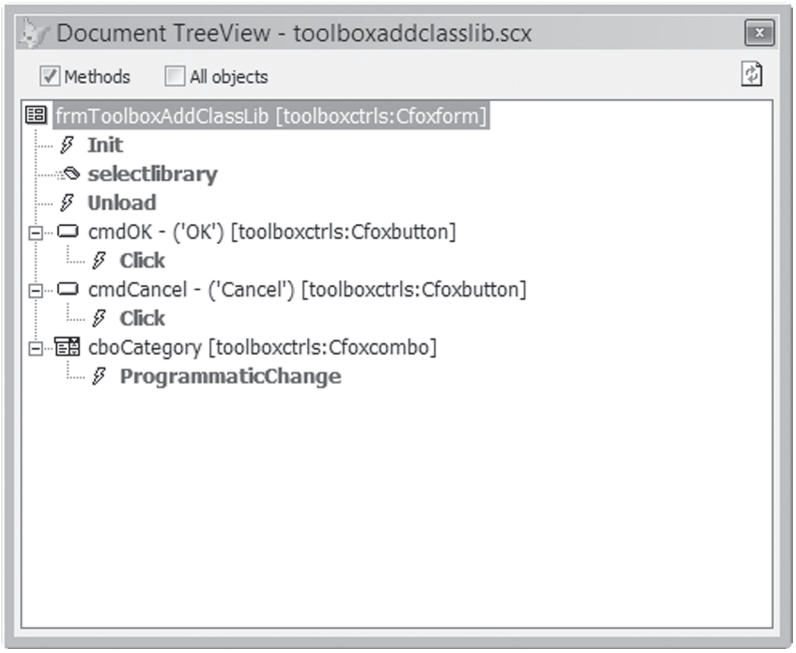

Figure 11. You can set Document Treeview to show only objects that contain code.

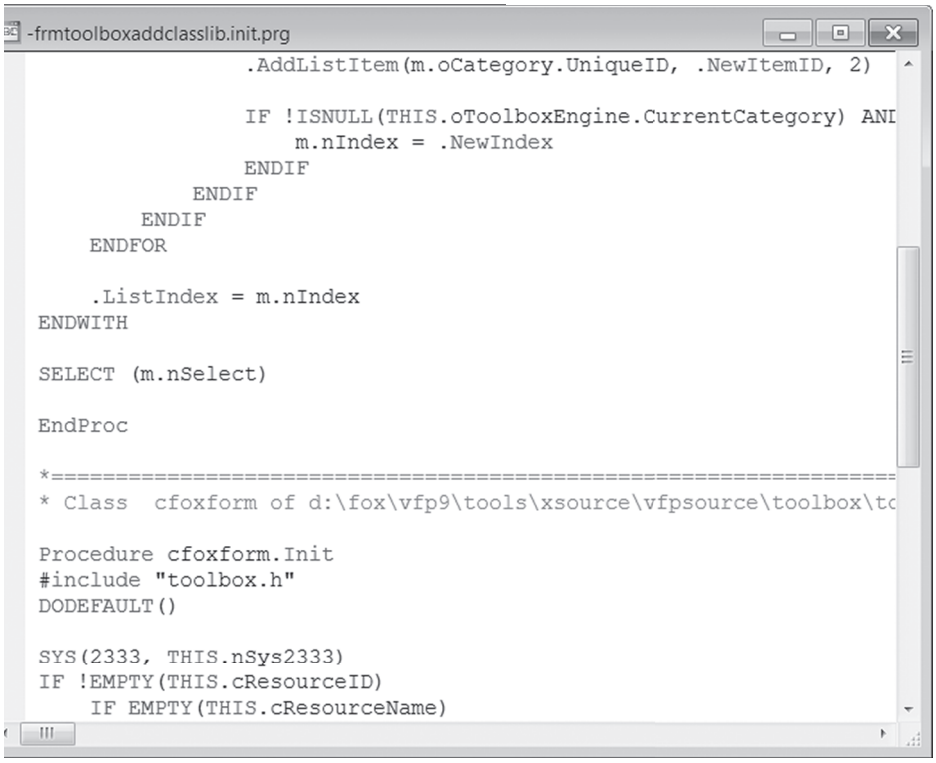

Figure 10. Using Ctrl+Click on a method in Method Treeview shows you all code for that method, from all levels of the inheritance hierarchy.

As you can see, this option makes it easy to see where there's code. Finally, if you uncheck Methods, the All objects checkbox is hidden, and you see all the objects in the form or class. Figure 12 shows Document Treeview for the ToolboxAddClassLib form with Methods unchecked.

You can also control the display using the tool's context menu. Figure 13 shows the basic menu, as it appears when you right-click on the background of the tool. (Right-clicking on a node, of course, includes options specific to the node.) The first three items below the divider in Figure 13 determine what information is included for an object. The first item, Show Caption, ControlSource indicates whether you want to include the Caption or ControlSource of an object in the display, to help you determine which object it is. In Figure 12, you can see, for example, that the Caption for the button cmdOK is 'OK.' As you'd expect, the Show class name and Show class library and name options are mutually exclusive;

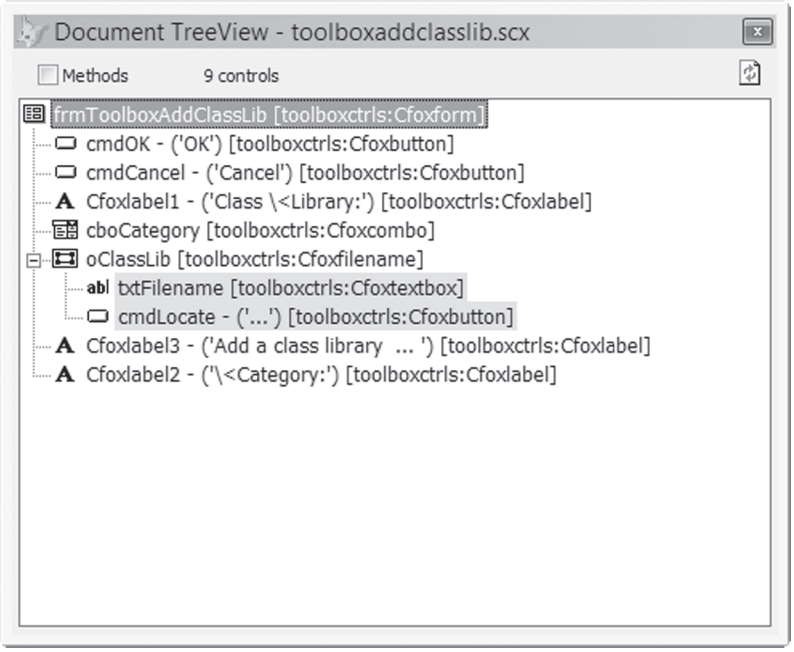

Figure 12. Unchecking the Methods checkbox tells Document Treeview to show all objects and no methods.

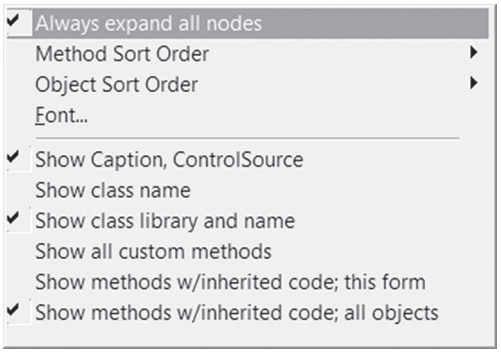

Figure 13. Document Treeview's context menu includes a number of ways to determine what's displayed.

checking one unchecks the other. However, you can choose to uncheck both and show no information about the class of an object.

The last three items in that section of the context menu determine which methods are displayed. By default, Document Treeview shows only methods that have code at this level of the hierarchy. Check Show all custom methods to also include all methods added to the current object. Check Show methods w/inherited code; this form to also show methods of the current object that have code higher in the inheritance hierarchy. Check Show methods w/inherited code; all objects to include methods of contained objects that have code somewhere in the inheritance hierarchy, as well.

The items above the divider control the appearance, but not the content of Document Treeview. The first item, Always expand all nodes, determines whether all items in the treeview are expanded when you open the tool or when you open a form or class with the tool open. The next two items determine the order in which methods and objects, respectively, appear in the list. For methods, the only choices are a case-sensitive alphabetical sort or a case-insensitive alphabetical sort. Objects offer a lot more choices, in addition to those two; the list is shown in Figure 14. You can leave them unsorted, in which case they appear in the same order as in the Property Sheet. You can also sort by TabIndex or from top to bottom or left to right.

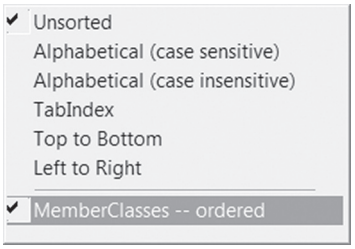

Figure 14. There are a number of ways to sort objects in Document Treeview.

By default, objects within controls that have the MemberClass property sort in creation order, the same order in which they appear in the Property Sheet. Checking the final option in Figure 14, MemberClasses -- ordered, sorts them according to the order specified, such as PageOrder or ColumnOrder.

Lots more to try

The tools described in this article and my last one are only a tiny subset of what Thor offers. Do yourself a favor: download and install Thor, and work your way through the menu to see what other gems you can find.

Author Profile

Tamar E. Granor, Ph.D. is the owner of Tomorrow's Solutions, LLC. She has developed and enhanced numerous Visual FoxPro applications for businesses and other organizations. Tamar is author or co-author of nearly a dozen books including the award winning Hacker's Guide to Visual FoxPro, Microsoft Office Automation with Visual FoxPro *and* Taming Visual FoxPro's SQL. *Her latest collaboration is* VFPX: Open Source Treasure for the VFP Developer. *Her books are available from Hentzenwerke Publishing (www.hentzenwerke.com). Tamar was a Microsoft Support Most Valuable Professional from the program's inception in 1993 until 2011. She is one of the organizers of the annual Southwest Fox conference. In 2007, Tamar received the Visual FoxPro Community Lifetime Achievement Award. You can reach her at tamar@thegranors.com or through www.tomorrowssolutionsllc.com.*