Introducing the Object and Collection Inspector

Debugging collections in VFP is difficult. This new tool, written in VFP, gives you a way to walk through object hierarchies, including collections.

Tamar E. Granor, Ph.D.

For the last four years, I've been working on a project that has grown to include a complex object hierarchy with many embedded collections. While VFP's collection class is quite useful, the debugging tools for collections are weak. In particular, there's no support for drilling down into collections. So I finally created my own tool.

In the process of building the tool, I learned some interesting lessons. In this article, I'll show you the tool I built; my next article will talk about the process of building the tool and some of the problems I encountered.

Why an Object and Collection Inspector?

I've been developing a project with an object hierarchy built from scalar objects and collections. That is, many of the objects in this hierarchy have properties that point to collections. The collections contain different objects, some of which have properties pointing to collections, and so on. (I described this project in the July, 2010 issue. I wrote about VFP's collections in general in the November, 2009 issue.)

From the beginning, debugging the collections was something of a pain. The VFP Debugger doesn't handle collections very well. While you can expand a collection object, doing so only shows the scalar properties, not the collection members.

To see an individual member of the collection in the Debugger, you have to enter the path to that member into the Watch window. While that's not a big deal if you have a single collection with a handful of members, when you're working with an object hierarchy, the inability to drill down to see what you have is big.

So, I decided to build a tool that would let me drill down into collections. My original name for the tool was Collection Inspector, but after working on it and with it for a while, it became apparent that it's more general than just a Collection Inspector; it's an Object Inspector, too. You hand it an object reference and it lets you drill into that object and whatever it contains, as deep as the hierarchy goes. (From here on out, I'll refer to the tool as the Object Inspector, though its title bar reads "Object and Collection Inspector.")

Designing the Object Inspector

As I started to design the tool, it seemed apparent to me that I'd want a treeview of some sort to represent the collection and its members and provide drilldown capability. I soon realized that I'd also want a way of showing data for each member of the collection. A little thought made it clear that what I really needed was the set of Explorer classes that Doug Hennig wrote about in the November, 2008, March, 2009 and May, 2009 issues.

I also quickly identified three kinds of things for which I wanted to show data: collections, other kinds of objects, and scalar values (which can also be members of collections).

In my next article, I'll cover various issues that arose as I learned to work with Doug's classes and figured out what I could and couldn't do in handling collections.

Using the Object Inspector

To use the Object Inspector, you call it with a reference to the object you want to inspect. Optionally, you can also pass the name of the object.

Figure 1 shows the Object Inspector as it opens for a collection called oCountries. The call that produced this display is shown in Listing 1; the code to create the collection is included in the downloads for this article as MakeCountryCollectionClasses.PRG.

Listing 1. This call to the Object Inspector produces the display in **Figure 1**.

```
DO Inspector.App WITH oCountries, "oCountries"
```

The whole point of the tool is the ability to drill into collections to see what's inside. When you do so, you get a list of all members. Each member is listed as Item[n] (where n is the item's index in the collection). If the item has a key, it's shown as well. Figure 2 shows the Object Inspector after expanding the oCountries collection.

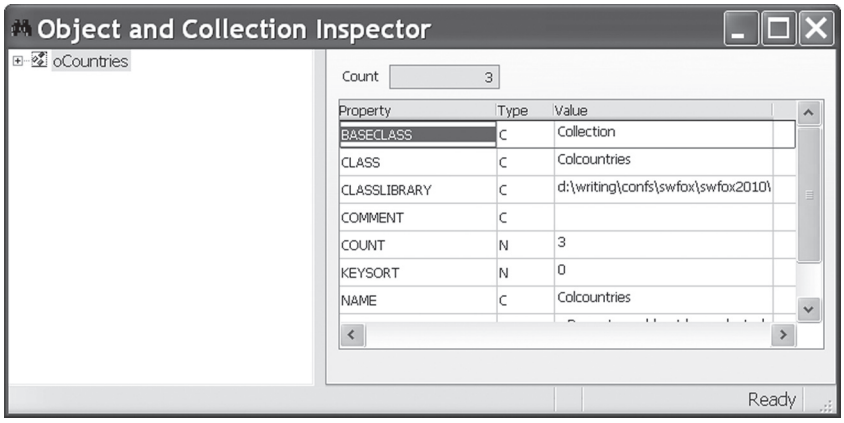

Figure 1. For a collection, the Object Inspector shows the count and then all the properties of the collection.

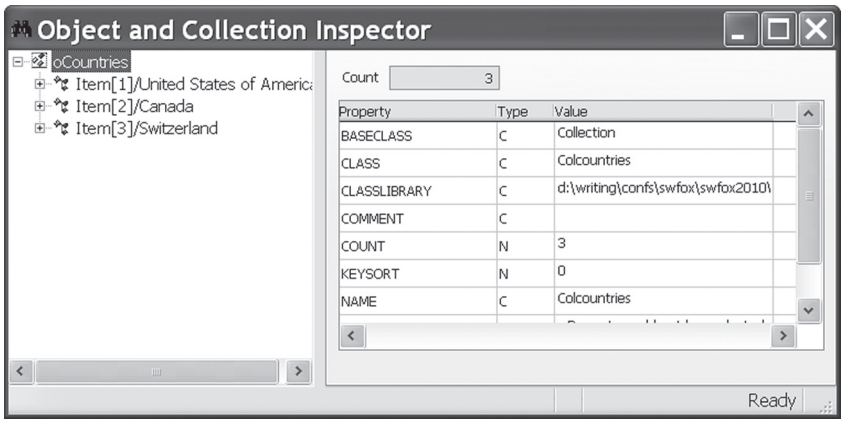

Figure 2. When you drill into a collection, each item is shown with its index number and, if it has one, its key.

When you click on a particular item, the right panel changes to show the details of that item. If it's an object, you see a list of properties and their values, as in Figure 3. When you drill into an object, as in the left pane of the same figure, you see any properties that reference other objects and collections, and you can drill into those things as well.

Collections can also contain scalar values, that is, items that are not objects. The Object Inspector handles those by showing the type and value in the right panel, as in Figure 4.

In some hierarchies, it's possible to have a cycle of pointers. For example, object A has a property that points to object B, and object B has a property that points to object A. Sometimes, the cycle is direct, like the A-B example. At other times, the cycle is indirect, with more than two objects involved. For example, in the countries hierarchy shown in the figures, a country object has an oStatesProvs collection containing state/province objects. The state/province object has an oCountry property pointing back to the country object.

These cycles present a problem for the Object Inspector. If the tool just allowed you to drill down with each pointer, you could end up doing so infinitely. That is, you might drill into Canada, expand the oStatesProvs collection, and then drill into Nova Scotia. You could then click that province's oCountry property, which would show the Canada node again. Then, you could drill into that node, expand the oStatesProvs collection and again drill into Nova Scotia (or any other province). And so forth and so on.

Not only does it not make sense to let a user do that, but it presented technical problems in setting up the data. Therefore, any given object is shown only once in the Object Inspector's hierarchy (the first time it's encountered when assembling the data). When you reach another pointer to the same object, the right

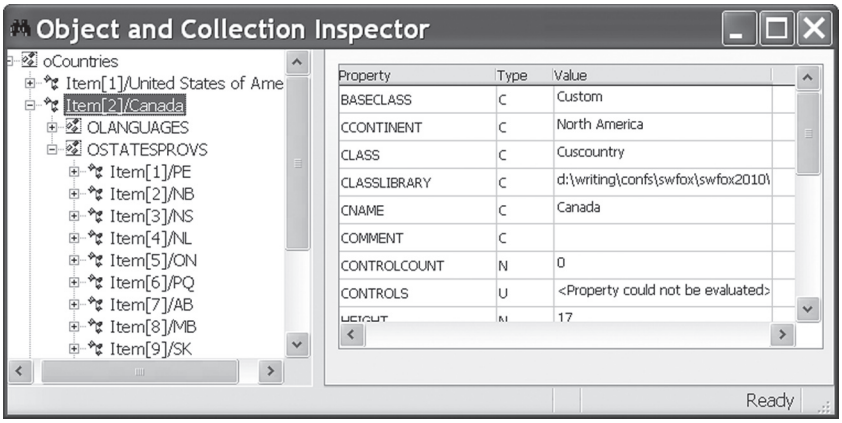

Figure 3. Click on a member of a collection to learn more about it. For objects, the right panel shows the object's properties and their current values. In this figure, the oStatesProvs collection inside the object has also been expanded.

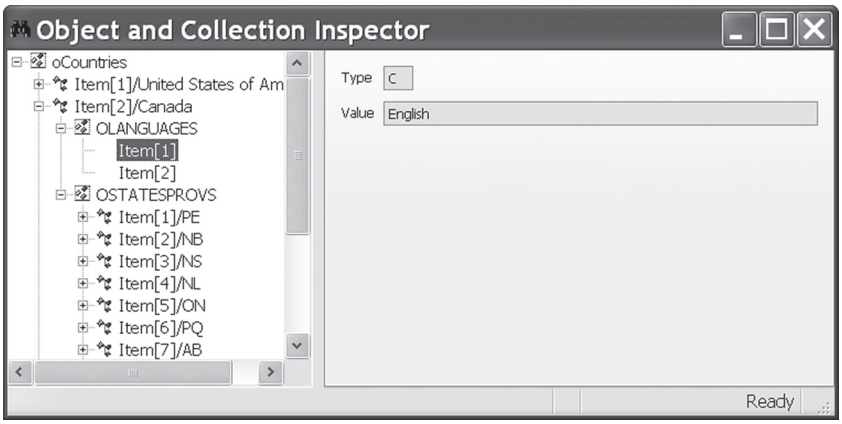

Figure 4. For scalar members of collections, the Inspector shows type and value.

panel tells you that the object appears elsewhere and gives you a link to return to it, as shown in Figure 5.

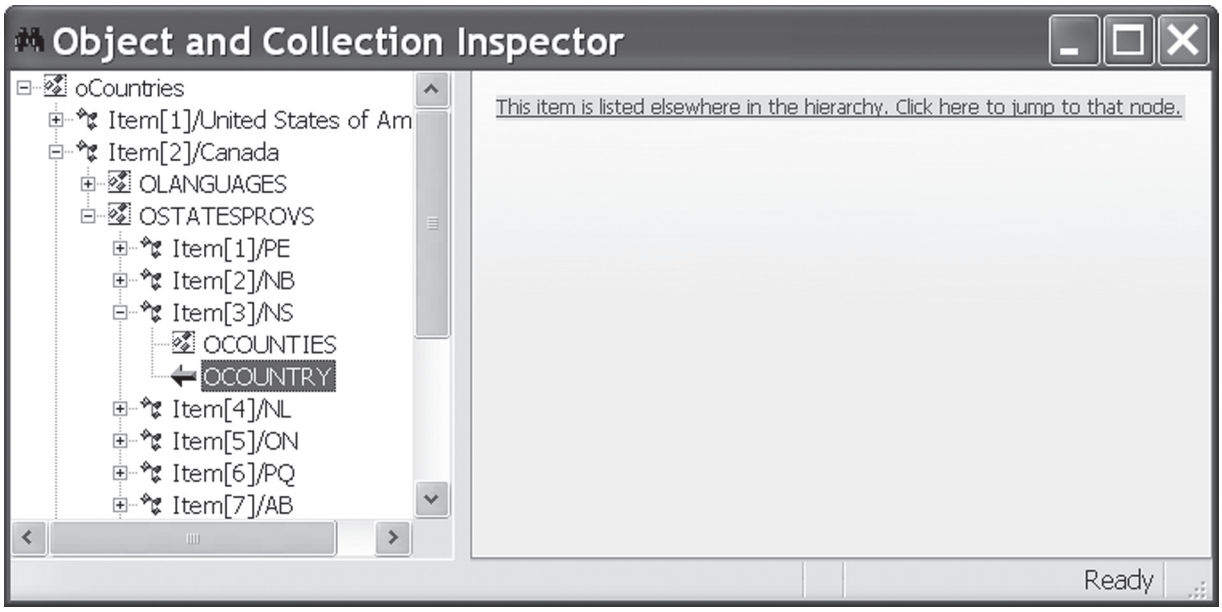

Figure 5. When a hierarchy contains cycles, only the first occurrence of an object appears in the right panel. Subsequent occurrences give you a link back to the first instance.

Some property values may be too long to display in the grid in the right panel (though the tool is resizable). You can double-click on any item in the Value column to open a Zoom window for that value. Figure 6 shows an example.

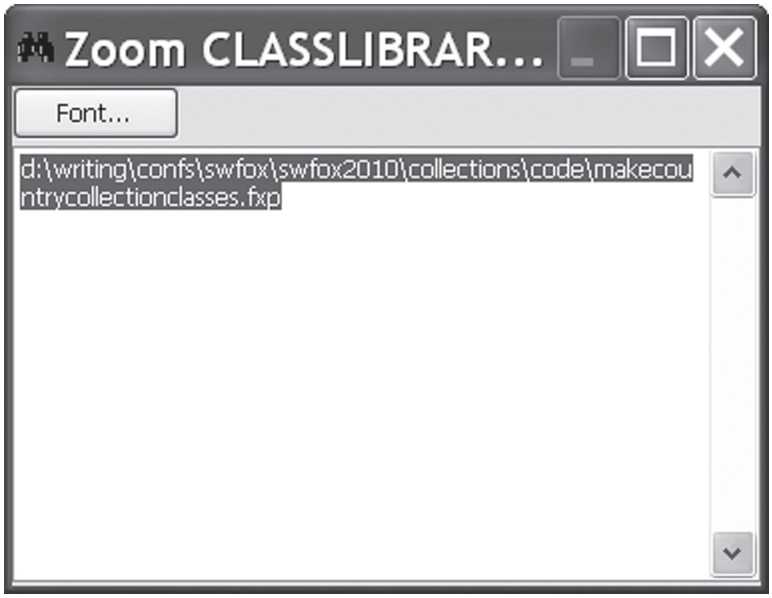

Figure 6. You can zoom the value of any property in the right-hand panel by double-clicking it.

More to come

The current version of the Object Inspector is included in this month's downloads. But the tool is a work in progress. The biggest item in planning stages is the ability to refresh the tool. Right now, it's a static display; that is, it shows the hierarchy at the time you run it. If you make changes to the objects involved while the Object Inspector is open, it does *not* update. A future version will give you the ability to update on demand.

Once you've tried the Object Inspector, please feel free to suggest additional enhancements, though of course I make no promises as to whether I'll add them.

I expect to submit the Object Inspector to VFPX when it gets a little farther along. At that point, additional help in developing the tool will be welcome.

Next time, I'll give you a look inside.

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. She currently focuses on working with other developers through consulting and subcontracting. 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 VisualFoxPro and Taming Visual FoxPro's SQL . Her latest collaboration is Making Sense of Sedna and SP2. Her books are available from Hentzenwerke Publishing (www.hentzenwerke.com). Tamar is a Microsoft Support Most Valuable Professional and 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.