July, 2001

Advisor Answers

Working with Collections

VFP 7.0/6.0/5.0/3.0

Q: I keep hearing about collections, but though I've been working with VFP since 1995, I don't understand how they work. Can you give me a simple overview?

–Dave Birley (via CompuServe.COM)

A: Think of a collection as the OOP version of an array. A collection contains objects in an indexed way. In VFP, the Forms property of _SCREEN is a collection, containing one entry for each form or custom toolbar currently defined. Each entry is an object reference to the form or toolbar. The Columns property of a grid is also a collection. It contains one entry for each column in the grid. So you can refer to the first open form in VFP as _SCREEN.Forms[1] and the third column of a grid called grdMyGrid sitting on the active form is _SCREEN.ActiveForm.grdMyGrid.Columns[3].

Once you have access to an object in this way, you can refer to its properties and methods just as you can with a more direct way of addressing. So you can change the caption of the first form, like this:

```
_SCREEN.Forms[1].Caption = "Hey, I'm first!"
```

or check a property of the column mentioned above, like this:

```
IF _SCREEN.ActiveForm.grdMyGrid.Columns[3].Sparse
   * do something
ENDIF
```

The collections that are built into VFP each have an associated …Count property. For example, FormCount tells how many items are in Forms, while ColumnCount indicates how many columns are in Columns. You can use these properties to find out how many there are or to control a loop:

```
IF _SCREEN.FormCount = 0
   * do something
ENDIF

FOR nColumn = 1 TO _SCREEN.ActiveForm.grdMyGrid.ColumnCount
   * do something to each column
```

```
ENDFOR
```

Although they look like arrays, you can't apply VFP's array functions to collections. So you can't issue something like:

```
nForms = ALEN(_SCREEN.Forms,1)
```

Of course, in this example, you can use FormCount, instead.

COM uses collections extensively. The Projects property of _VFP is a COM collection, for example, as is the Files collection individual projects contain. When automating other applications, collections tend to play a big part, too. For example, Word's object model includes Documents, Paragraphs, Sections and Styles collections, among many others. Outlook's object model has a Folders collection; the individual Folder objects often contain another Folders collection. In the Windows Scripting Host (WSH), there's a Drives collection, a Subfolders collection, and a Files collection.

Most COM collections have a Count property and an Items property (though the WSH calls it Item rather than Items). The Count property tells you how many items are in the collection. The Items property gives you access to the individual members of the collection. For example, you can reference the first open project in VFP by using:

```
_VFP.Projects.Items[1]
```

Either parentheses or square brackets are acceptable here. If you've created an instance of Word and stored a reference to it in oWord, you can access the third open document in Word as:

```
oWord.Documents.Items[3]
```

Many collections let you leave out the Items keyword and just reference their members directly from the collection. So you can get to that same project with:

```
_VFP.Projects[1]
```

and to the same document as:

```
oWord.Documents[3]
```

It's not unusual for a collection to let you reference its members both numerically (as in the examples above) and by name. So, if it's open, I can access the TasTrade project as:

```
_VFP.Projects["TasTrade.PJX"]
```

If the third document is "MyDoc.Doc", I can access it as:

```
oWord.Documents["MyDoc.Doc"]
```

The WSH is different – it doesn't accept a numerical index, but requires the name of the object.

Many object models include one or more properties to provide direct access to collection members that are currently in use. For example, _VFP has an ActiveProject property and Word has an ActiveDocument property.

As with VFP's native collections, once you have a reference to a particular member, you can check or change properties or call methods:

```
_VFP.Projects["TasTrade.PJX"].Build()
```

The technique for adding members to a collection varies with the collection. Many have an Add method that lets you put something else in. For example, to open a new document in Word and add it to the Documents collection, you use:

```
oWord.Documents.Add()
```

In other cases, though, the collection gets a new member because of another action. For example, opening a VFP project with MODIFY PROJECT adds it to the Projects collection. There's no way to directly add a project to the collection. Generally, it's the nature of the objects in the collection that determines whether there's an Add method or not.

Removing items from a collection also varies. In some cases, objects are removed when another action occurs. For example, a project is removed from the Projects collection when you close it. In other cases, there's a method to remove the object. In general, when such a method exists, it's a method of the object itself, not of the collection. Sometimes, that method has another action that results in the object being removed from the collection. For example, when you execute a Document's Close method, it's removed from the Documents collection:

```
oWord.ActiveDocument.Close()
```

Finally, in some cases, there is actually a Remove method. For example, you can remove a file from a project's Files collection (which has the effect of removing it from the project) using the File object's Remove method:

```
_VFP.ActiveProject.Files["Junk.Txt"].Remove()
```

One of the advantages of a collection is that it makes it easy to process all the members in some way. The FOR EACH loop is the easiest way to do something to every member of a collection. For example, to see what projects are open, you can use:

```
FOR EACH oProject IN _VFP.Projects
   ?oProject.Name
ENDFOR
```

Obviously, what goes inside the loop can be much more complex, and can check some property of the object to determine whether to process it. For example, this loop checks each open document in Word and, if it's already been saved, closes it:

```
FOR EACH oDocument IN oWord.Documents
   IF oDocument.Saved
      oDocument.Close()
   ENDIF
ENDFOR
```

Understanding collections is a key part of working in the COM world. Since VFP has native collections as well, getting a firm grasp on collection fundamentals can improve your productivity.

–Tamar