Issue Date: FoxTalk May 1998

Automation from a VFP Perspective

John V. Petersen

Last month, John introduced you to the basics of ActiveX Automation. This month, he expands on the topic by discussing the various issues that face you -- the Visual FoxPro developer who wishes to include ActiveX Automation in your applications.

There are many examples of ActiveX Automation available today in books and on the Web. Unfortunately, these examples are written from the perspective of the Visual Basic developer. This probably shouldn't come as a big surprise, since ActiveX Automation is predicated on the Visual Basic for Applications (VBA) language. If you're developing in Visual Basic, your work is relatively painless in that you can copy and paste VBA code directly in your VB application. In VFP, however, the effort involved in getting VBA code to work is a bit more extensive. But take heart -- while initially frustrating, VBA code samples can be made to work in VFP. You just need to understand how to make the modifications to the code.

A quick word on the macro recorder...
Before going further, it's important to mention the macro recorder that's present in the applications that make up Microsoft Office. Whether you use Word, Excel, or PowerPoint, code can be automatically generated by recording tasks that you'd perform with the mouse and keyboard. If you're currently employing automation and aren't using the macro recorder, you're missing out on a wonderfully productive tool that can seriously reduce your workload. In addition to generating code, the macro recorder is a great teaching tool. Last month, I mentioned that understanding the object model of an automation server application is the key to making your automation efforts successful. After all, knowing where to find something is often more complicated than understanding how to do something. The macro recorder can quickly de-mystify a server's object model.

If you haven't explored the macro recorder, try this simple example using Word 97:

1. Open a Word document.
2. Click the Tools menu.
3. Select the Macro menu.
4. Select the Record New Macro option.

The dialog box should look similar to **Figure 1**.

Figure 1. In the Record Macro dialog box, you can specify the name of your macro, where it will be stored, and a description of what the macro does.

When creating a macro, it's important to understand the scope of documents to which your macro will apply. By default, the macro dialog box will point to the template upon which your document was created. This might or might not be what you want. Alternatively, you can assign your macro to the current document by changing the selection in the Store Macro In drop-down list box. In the example shown, I've specified that my print macro will be stored with the current document. This will be important, as I'll need to know where to find the macro in the Visual Basic Editor.

Once you press the OK button in the Record Macro dialog box, a toolbar will appear in Word with two buttons. **Figure 2** shows the toolbar that will appear in your document. The first button will stop macro recording, and the second button will pause it. The following steps will complete the macro:

1. Select the File menu.
2. Select Print.
3. Select a printer and press OK.
4. Select the Stop Recording Macro button in the Macro Recorder toolbar.

Figure 2. The Macro Recorder toolbar can both stop and pause macro recording.

That's it! You've generated VBA code. Now, you need to look at what's been generated. There are two ways to do this. The first is to go back to the Tools menu, choose Macro, and then select the Macros option. The available macros are displayed in a dialog box, as illustrated in **Figure 3**.

Figure 3. The Macros dialog box lists the macros associated with the document selected in the drop-down combo box.

It's important to note that if you have multiple documents open, you might not see your macro listed. In this example, the name of the document is Document4. So, to be sure I see the new macro, I need to select the Document4 window, making it the active document in Word. Once I've done this and displayed the Macros dialog box, I can then press the Edit button. Pressing the Edit button brings forth the Visual Basic Editor. This is illustrated in **Figure 4**.

Figure 4. The Visual Basic Editor in VBA is identical to the code window in the full Visual Basic Development System.

One thing that makes automation a challenge is the lack of feedback the developer gets when invalid arguments are passed. Usually, the error consists of an error message stating that some sort of data type mismatch occurred. Other times, it might be a generic OLE error message. Regardless of which message you receive, it's usually of little or no help in debugging the situation. This is why an understanding of how VFP treats automation servers differently than VB is important.

Here's a good example. If you attempt to copy and paste this code in VFP, it will bomb. For one thing, the macro recorder assembled the arguments that are passed to the PrintOut Method in a non-default order. In FoxPro, you can't simply toss arguments to a function in any old order, but in the VB environment, this is okay. The reason is that VB supports a mechanism called "named arguments."

The VBA code -- up close and personal
The following is the code that was generated by the macro recorder:

```
Sub myprintmacro()
'
' myprintmacro Macro
' Macro recorded 03/10/98 by John Viktor Petersen
'
  ActivePrinter = "HP DeskJet 550C"

  Application.PrintOut FileName:="", _
   Range:=wdPrintAllDocument, Item:= _
   wdPrintDocumentContent, Copies:=1, _
   Pages:="", PageType:=wdPrintAllPages, _
   Collate:=True, Background:=True, PrintToFile:=False
End Sub
```

The first line of code is pretty straightforward in that the ActivePrinter property is being assigned the name of a printer. In the second line, the PrintOut method of the Application object is being invoked. As I mentioned, one of the nice features of VBA -- and VB, for that matter -- is that named arguments are supported. In other words, arguments in VBA can be recognized regardless of the order in which they're passed. The benefit to this is that some arguments can be omitted, and there's no preset order in which the arguments have to be passed. You can accomplish the same thing in VFP by parsing strings, but it's a great deal more work.

Of course, you can omit the naming of parameters, but in doing so, you must pass the arguments in their default order, and you must provide placeholders for those arguments that separate arguments you wish to pass. For example, suppose you wish to specify a filename and the number of copies to be printed. You must specify the defaults for the range and item parameters as well. In VFP, however, knowing the order and quantity of arguments is necessary to get things working properly.

Don't judge a book by its cover...
Thus, before porting the VBA code to VFP, three things must be ascertained:

1. The number and order of arguments the method expects
2. The data type of the arguments
3. Values of constants, if any

At first glance, it would appear the PrintOut method takes nine arguments. In fact, this isn't true. In last month's column, I introduced you to the Object Browser in VBA. **Figure 5** shows the Object Browser with the PrintOut method highlighted.

Figure 5. For methods, the Object Browser can indicate the arguments a method can accept.

Are you sitting down? Incredibly, the PrintOut method expects up to 15 arguments. Also, according to the Object Browser, by default, the default and append arguments should be the first and second arguments passed, respectively. Upon observing the code, the macro recorder had something different in mind -- instead, it made the filename first. The problem is that the filename argument is a string, but the default argument -- background -- is Boolean. Because VFP must pass arguments in the default order, a data type mismatch error will occur.

Another feature of the VB environment is the ability to intrinsically work with constants. While you can define your own constants in VB, many are already predefined. The Object Browser can help here as well, since these constant definitions are contained in the server type library. This capability is illustrated in **Figure 6**.

Figure 6. The Object Browser can also indicate the value of intrinsic constants used in VB/VBA.

In this example, all of the constants -- wdPrintAllDocument, wdPrintDocumentContent, and wdPrintAllPages -- have a value of 0.

Porting the code to Visual FoxPro

The following code is a VFP program that illustrates how the Word-produced VBA code can be made VFP-compatible:

```
* printinword.prg
#Define True .T.
#Define False .F.
#Define wdPrintAllDocument 0
#Define wdPrintDocumentContent 0
#Define wdPrintAllPages 0

oWord = CreateObject("word.application")
oWord.Documents.Add
oWord.ActivePrinter = "HP DeskJet 550C"

* This attempt would bomb because the macro
* recorder did not assemble the arguments
* in the default order. In this case,
* the macro recorder made the filename
* parameter first. By default, the
* background argument is first.

*oword.PrintOut("",wdPrintAllDocument,;
               wdPrintDocumentContent,;
               1,"",wdPrintAllPages, ;
               True,True,False)

* The printout method with
* no arguments will invoke the defaults.

oword.printout

* Placeholders must be used since VFP does not support
* named argument passing. The number of copies is the
* eighth parameter. In VB, the syntax is simplified with
* the use of named arguments: oword.printout Copies:=1

oWord.PrintOut(True,False,wdPrintAllDocument,;
               "","1","999",wdPrintdocumentcontent,2)

* Or, if the arguments before the number
* of copies argument can be kept at their default,
* the following will work as well.

oWord.PrintOut(,,,,,,,2)
```

One tip that I've found to save work is to define the word True as .T. and False as .F.. If you don't do this, you'll find yourself doing lots of searching and replacing. Also, I define any constants in VFP exactly as they appear in VB. This helps in matching up the VFP calls with the VB-based documentation.

In this last example, I elected to print two copies of the document. This requires that placeholders for each preceding argument must be passed as well. The last call in the VFP code will work, but it's much less readable. Do yourself (and anybody else who needs to work with your code) a favor -- explicitly pass the defaults. It will make copying and pasting VBA code much easier.

One other subtle difference in the VFP version of the code deals with parentheses. In VB, method calls can be made and arguments can be passed without the use of parentheses. In fact, unless the method is returning a value, VB doesn't allow you to use parentheses. In VFP, on the other hand, if arguments are being passed to a method, parentheses must be used.

Finally, inside the VBA environment, properties -- which are members of the application -- can be referred to directly. In the VBA code, the ActivePrinter property could have been referred to as Application.ActivePrinter. The same can't be said for methods, as they must be fully qualified. While the macro recorder is nice, it can be inconsistent. Whenever possible, fully qualify the properties in your VBA code. It will make the copying and pasting operations that much more simple.

Summary

VBA code in VFP works! It just takes an understanding of both the VBA code itself and how VFP can make use of the code. With an understanding of the subtle differences between the language constructs of VB and VFP, and an understanding of the server object model, VFP can make use of the code with a few modifications. Also, by making use of the macro recorder and the Object Browser, your knowledge of a server's object model -- such as Word -- can be greatly accelerated.

Next month, I'll extend things by discussing scenarios in which to include your automation code and why you might want to extend your applications with automation. Finally, many of you might remember the old Dr. FoxPro's Answer Clinic column and its Q&A format. I'd like to print one or two of your questions each month regarding automation and VFP. So, if you have a nagging automation problem you'd like to see addressed, send me your question and it might show up in a future issue of *FoxTalk!*