Chapter 2
The Office Servers

Learning about the Office servers is a challenging task, but there are a variety of approaches, tools, and resources available. Certain features of the Office applications, like collections, are common to all of them—mastering them up front will pay off fast.

Microsoft Office incorporates a whole bunch of applications and applets, but the main reason people buy and use it are the big two: Word and Excel. Along with them come a couple of others that are used by an increasing number of people who may not have planned to do so initially—PowerPoint and Outlook.

Word is an incredibly powerful word processor. It provides users with the ability to create anything from a simple note to complex reports and legal documents, and everything in between.

Excel is a sophisticated spreadsheet program. It includes integrated graphing capabilities and has a significant collection of built-in financial, statistical, and mathematical functions.

PowerPoint builds and displays presentations. Like the other applications, its abilities range from simple to complex—it offers everything from straightforward slides with a few bullet points and some text to full multimedia shows.

Outlook is harder to describe because it has a little of everything. For some, it's simply a mail client. For others, it's a calendar and to-do list. It also incorporates an address book, a journal, a notebook, and lots of opportunities to customize.

Exploring the Office servers

All of the Office applications share a programming language—Visual Basic for Applications, or VBA. For Automation developers, having a shared programming language isn't as impressive as it sounds because, while each Office application uses the same syntax and commands, that's only helpful if you're actually writing in that programming language. For Automation, you're not. You're writing in the language of the client application—the one controlling the Automation process.

On the other hand, for a variety of reasons, it's very handy to be able to read VBA when you're writing Automation code. Since we're all programmers here, that's not a big problem. But there are a couple of peculiarities about VBA (from the FoxPro developer's perspective, anyway) that make it hard to read for those who are unfamiliar with it. We'll look at each of these issues as we dig into the Office servers.

To use the Office servers, then, the key issue is finding out what methods and properties they have and determining the parameters to pass to their methods. There are three main approaches to take. In most cases, you'll need to combine all three to get what you need. Beyond those, there are a couple of other ways you can learn about Automation and then test out what you've learned.

Read the fine manual (RTFM)

Each of the Office servers has a Help file that documents its members. The method of getting to that Help file varies with the application. In the big three—Word, Excel, and PowerPoint—you can access it from the main Help menu for the application. On the Contents page, go down almost to the bottom. In Word and Excel 2000, look for "Programming Information" just above the bottom of the Contents list. In PowerPoint 2000, the entry to find is "Microsoft PowerPoint Visual Basic Reference," also near the bottom of the Contents list. In all three cases, you can open the specified item to see a list of Automation topics. Outlook's Automation Help is well hidden. You have to open the "Advanced Customization" topic to find "Microsoft Outlook Visual Basic Reference."

Getting to the Help files is a little different in Office 97. You start out the same way in Word, Excel, and PowerPoint—by choosing the Contents page from Help and scrolling down to the bottom. Then, in all three, look for "Microsoft X Visual Basic Reference," where X is the product you're using. When you choose that item, it opens to reveal "Visual Basic Reference." Choosing that item opens a new Help window that contains just the Automation Help for that product. Getting to Visual Basic Help for Outlook 97 is extremely complex. The steps are spelled out in the main Help file—search for "Visual Basic." Understand, though, that VBA is not the primary programming language for Outlook 97, and automating it with VBA is trickier in some ways than automating the other Office 97 products.

If you can't find the appropriate item in the Help contents, it means you didn't install the VBA Help file with the application. To do so, you have to choose Custom installation. The standard installation doesn't include these files. (In Office 97, the standard installation was called "complete," but it wasn't.) You can install the VBA Help files at any time by running Setup again. In Office 2000, you do so by choosing Visual Basic Help from Office Tools. (By default, the VBA Help files are set to install on first use. If you know you're planning to do Automation work, we suggest that you go ahead and install them on your hard drive in the first place. The whole set of four [Word, Excel, PowerPoint, and Outlook] for Office 2000 is less than 5 MB.)

By this point, you may have noticed that getting to the VBA portion of the Help can be something of a pain. In Office 2000, VBA Help is integrated into the regular Help and doesn't open a separate window, so working within it is difficult, too. In Office 97, once you get to VBA Help, it's way too easy to close it—just hit ESC and it's gone.

Fortunately, there's an easy alternative here. Although accessible from within the main Help file for its respective application, the VBA Help for each Office application actually lives in a separate file. In Office 97, the filename is in the form VBAapp8.HLP, where "app" is some form of the application name. For Office 2000, it's VBAapp9.CHM, because the Office 2000 applications use HTML Help; that's why you can't close Help with ESC anymore. Given all of these difficulties in getting to VBA Help, we strongly recommend that you create shortcuts on your desktop for each of the Help files you're likely to use. Then, they're only a double-click away.

For the most part, the upgrade from Office 97 to Office 2000 is either positive or neutral. We also like HTML Help in general. In fact, the electronic version of this book uses HTML Help. However, the Office 2000 Help files, including the VBA Help files, are far less useful than their Office 97 equivalents. In Office 97, Help files had three pages: Contents, Index, and Find. Index featured an alphabetic list of terms from the Help file (like the index of a book), while Find provided full-text searching. For the VBA Help files, the Index page offered a quick way to go directly to any object, property, or method.

Office 2000 Help files have only two pages: Contents and Search. The team at Microsoft merged the Index and Find pages to create a single Search page, which offers full-text search without the real-time component of the Find page. There is no way to move quickly to a keyword. We suspect that there are some good usability arguments for the new search mechanism in the Help files for the main products. End users probably find this approach more productive. However, we cannot understand how programmers can be expected to use the VBA Help files productively without an index. (The best substitute for the Index page is the separate alphabetical lists of Objects, Properties and Methods on the Contents page. While we kept the VBA Help files on the Index page in Office 97, in Office 2000, we tend to keep it set to the Contents page.)

Each of the VBA Help files includes a diagram of the object model for that product. The model is "live"—when you click on it, you either move to another level of the diagram or to the appropriate entry in Help. **Figure 1** shows the top level of the Word object model diagram.

The Help entries for objects include small pieces of the object model diagram as well—clicking on these also jumps to the indicated entries. **Figure 2** shows the entry for the Documents collection in the Word VBA Help file. The dotted box that says "Multiple Objects" has been clicked, bringing up a dialog box of objects contained in the Document object.

Despite the difficulties in getting directly to a particular item in Office 2000 Help (a task that was incredibly easy in Office 97), the VBA Help files are generally clear and correct. If you know what you're looking for, you can find it there. The real challenge, then, is to figure out what you're looking for.

Let the server write the code

Word, Excel, and PowerPoint have macro recorders that let you turn user actions into VBA code. (Outlook supports macros, but unfortunately it doesn't include a macro recorder.)

So one way to figure out how to automate something is to record a macro to do it, and then examine the macro.

From the menu, choose Tools|Macro|Record New Macro to start recording. Give the macro a name (preferably a meaningful one, so you can find it again and so you'll know what it is when you come across it six months from now). Then interactively perform the operation you want to automate. When you're done, click on the Stop Recording button on the Macro toolbar (which appears automatically when you start recording).

***Figure 1**. The Word object model diagram. The Help file for each Office product contains a diagram of the object model for that product. Clicking on one of the rectangles takes you to the Help entry for that object. Clicking an arrow takes you to another level in the object model diagram.*

To look at the recorded macro, choose Tools|Macro|Macros from the menu; that opens the Macros dialog shown in **Figure 3**. Highlight the macro you just created (see, you need to know the name already), and then choose Edit. This brings you into the Visual Basic Editor (VBE)—**Figure 4** shows the highlighted macro from Figure 3 as it appears in the VBE. This particular macro was recorded in Word. (You can also get to the VBE by choosing Tools|Macro|Visual Basic Editor, or by pressing Alt-F11.)

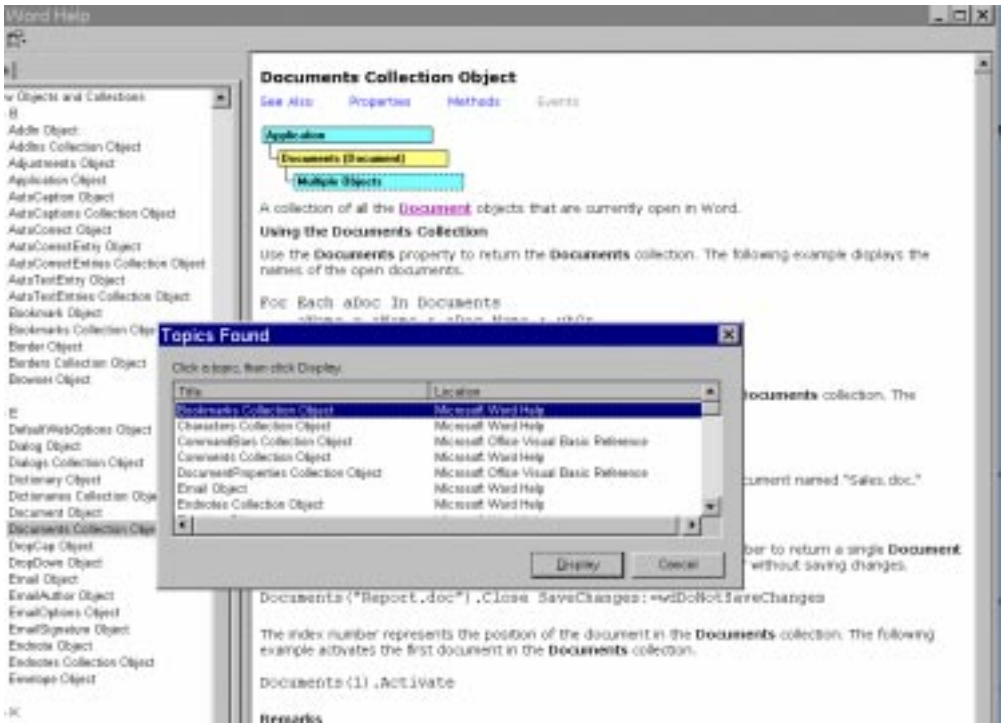

***Figure 2**. Live Help. Even within Help entries, the object model diagrams are live. In this case, clicking on the "Multiple Objects" rectangle calls up a dialog that lists other objects contained in the Document object.*

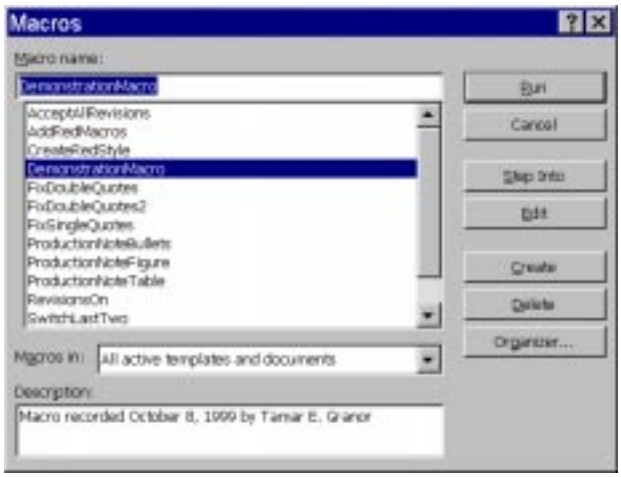

***Figure 3**. The Macros dialog. This dialog lists all the macros you've recorded or otherwise stored in an Office application. It's one entry point to the Visual Basic Editor.*

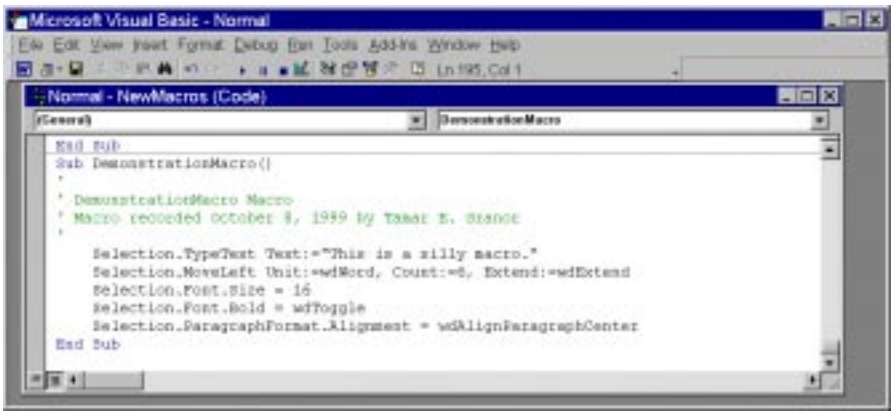

Figure 4*. Viewing macros. Editing a macro takes you to the Visual Basic Editor. At first glance, the code may seem mysterious, but just a few tricks can decode it.*

Converting a VBA macro to VFP code is harder than it should be for several reasons. The macro shown in Figure 4 demonstrates all of them. Consider this line of Word VBA code as you read the following sections. The line moves the insertion point (the vertical bar that determines where the next character is inserted) six words to the left, highlighting the words in the process:

```
Selection.MoveLeft Unit:=wdWord, Count:=6, Extend:=wdExtend
```

Default objects

First, unlike VFP, VBA makes some assumptions about what object you're talking to. In the preceding line, Selection translates to VFP's This.Selection, which represents the current selection (highlighted area) of the Word instance.

In each Office application, certain objects are considered default objects in the VBA environment. Your code won't be treated so kindly—you need to be explicit about what object you're addressing. The code you send to the Automation server must be addressed to the correct object.

Named parameters

VBA allows methods to use what are called *named parameters*. In the example code line, the method called is MoveLeft. Three parameters are passed, each in the following form:

```
parameter name := parameter value
```

This syntax allows VBA programmers to include only the parameters they need and not worry about the order of the parameters. Since some VBA methods have a dozen or more parameters, this is a very handy option.

However, VFP doesn't support named parameters; you must specify parameters in the proper order. Fortunately, the Help files show the parameters in their required order. (That wasn't true in versions of Office before Office 97; Help for many methods showed the parameters out of order, and finding the correct order was extremely difficult.)

When translating from VBA to VFP, add parentheses around the list of parameters, and delete the parameter names and ":=" symbols. Check Help (or the Object Browser, discussed later in this chapter) to determine the correct order and number of parameters. Usually, the macro recorder puts the parameters in the correct order; however, some parameters may be omitted, as named parameters allow VBA developers to leave out any parameters that are to take the default value. Be sure to check Help for omitted values. Also check to ensure that the macro recorder really did put them in the proper order; occasionally it doesn't.

Defined constants

The first parameter in the example line shows the third problem that occurs in translating VBA to VFP. It specifies that a parameter called Unit should have the value wdWord. But what is wdWord? It's one of thousands of defined constants available in Word's version of VBA. (It turns out that wdWord is 2.)

The VBA Help files don't supply the values of defined constants. In fact, Help uses them exclusively and doesn't show their actual values anywhere (ditto for the macro recorder). (Outlook is the exception here. It has a Help topic titled "Microsoft Outlook Constants" that includes a complete list.) To find out what wdWord and all of the others stand for, use the Object Browser available through the Visual Basic Editor. (See the section "Take me for a browse" later in this chapter.)

If there was ever a reason to use header files, VBA constants is it. However, to build the header file, you need to find the values of the constants. Rick Strahl of West Wind Technologies has created a freeware tool that reads a COM type library, extracts the constants, and creates a Visual FoxPro header file. The tool, called GetConstants, is included in the Developer Download files available at www.hentzenwerke.com and can also be downloaded from Rick's site (www.west-wind.com). The Developer Download files also include header files for each of the Office applications.

We do *not* recommend using these files as is in your Automation work. The number of constants contained in them is mind-boggling. The smallest set is for Outlook—it contains 251 constant definitions. Word's file is nearly 10 times that size, with more than 2,400 constants defined. On one of our test machines, saving a form with no controls, but pointing to the full Word constant definition file as its Include file, took more than five seconds. (By contrast, on the same machine, saving a totally empty form took well under a second.)

However, having the complete set of constants at your disposal is very handy. You can cut and paste from them to create header files appropriate to the tasks you're doing. Rick's tool also makes it easy to keep your header files up-to-date as Microsoft adds new constants.

Macro recorder tips

Before moving on to the Object Browser, there are a couple of things worth noting about the macro recorder and the Visual Basic Editor. First, be aware that the macro recorder doesn't always produce the *best* code for a task. The code it produces gives you an idea of what can be done, but there are often better ways to accomplish the same task.

The macro recorder is focused on interactive users. While it doesn't just record the user's keystrokes, neither does it have the intelligence to figure out the task at hand and put together a complete, logical program to do it. Lines and lines of code are generated. A good deal of the time, several lines of code can be replaced with a single method call. Other times, many lines of code are generated that set every possible property for an object. Perhaps 15 properties are set when you only changed one. Many times, you won't need to set all of them. However, when you're trying to figure out which properties need to be set, be sure to consider that many users change the application's defaults, so don't assume anything!

Second, the Visual Basic Editor has a feature called IntelliSense that makes writing code there easier. When you type an object name and a period (like "oRange."), an appropriate list of properties and methods appears. When you choose a method from the list (use Tab to choose without ending the line), a list of parameters for that method appears like a ToolTip to guide you. As you enter parameters, your position in the parameter list is highlighted to keep you on track. This can be very handy when you're trying to figure out what parameters to pass to a method. Unfortunately, at this writing, Visual FoxPro doesn't natively support IntelliSense (though early demos of VFP 7 include it). Write the code in the VBE, and cut and paste it into VFP.

Take me for a browse

One of the most powerful tools available for figuring out Automation code is the Object Browser (see **Figure 5**). It lets you drill into the various objects in the hierarchy to determine their properties and methods, see the parameters for methods, determine the values of constants, and more.

The easiest way to find out about a specific item is to type it into the search dropdown, then press Enter or click the Find (binoculars) button. The middle pane fills with potential matches. Choose one to learn more about it in the main section of the Browser underneath. The left pane is filled with the properties, methods, collections, and constants. The right pane describes what's available for the highlighted item in the left pane. In Figure 5, the Object Browser has been used to determine the value of the constant wdWord. In the bottom-most pane, you can see that it's a constant with a value of 2.

The Object Browser is also useful for moving around the object hierarchy to get a feel for what the various objects can do. **Figure 6** shows the Object Browser with Excel's objects rather than Word's (the Visual Basic Editor was opened from Excel). The members of Excel's Workbook object are shown in the right pane. The PrintOut method is highlighted, so the very bottom panel shows its (complex) calling syntax. The advantage of this approach over Help is that the Object Browser actually looks at the type library, so the list it shows is more likely to be correct than Help. Even better, the Object Browser and Help can work together. Press F1 in the Browser, and Help opens to the highlighted item.

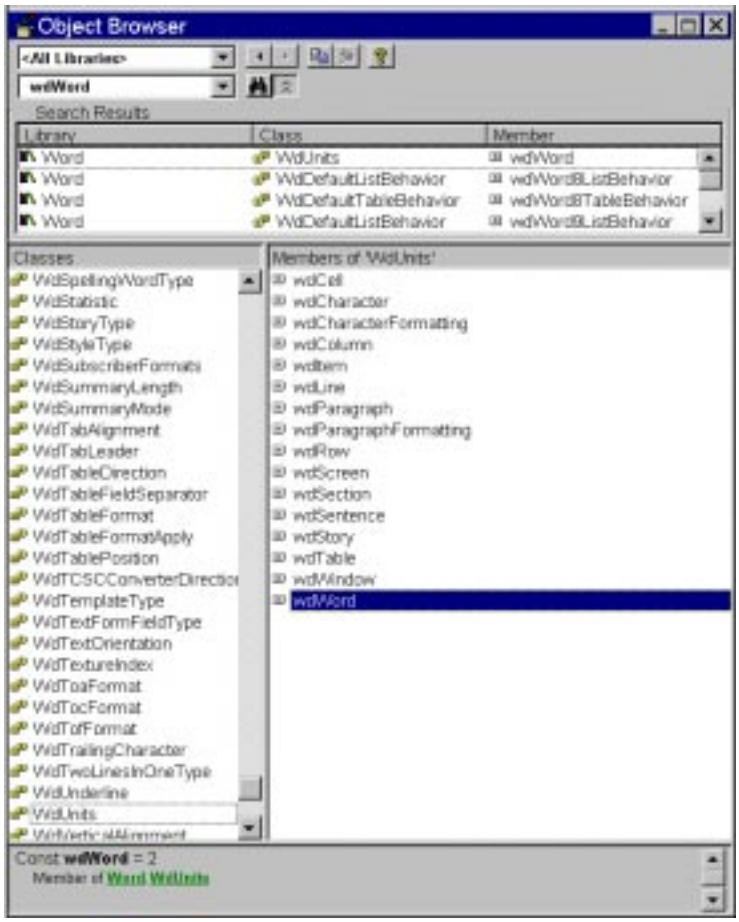

***Figure 5**. The Object Browser. This powerful tool lets you drill down into objects, find out constant values, and determine parameters. Here it shows that wdWord is a constant, is a member of a group of constants called WdUnits, and has a value of 2.*

The Browser is also useful for exploring the object hierarchy itself. **Figure 7** shows the PowerPoint version of the Object Browser (this time, the VBE was opened from PowerPoint). The Presentation object's members are shown in the right pane. The Slides property is highlighted. In the bottom pane, we learn that Slides is a reference to a Slides collection. Clicking on the underlined Slides takes us to the Slides collection, shown in **Figure 8**.

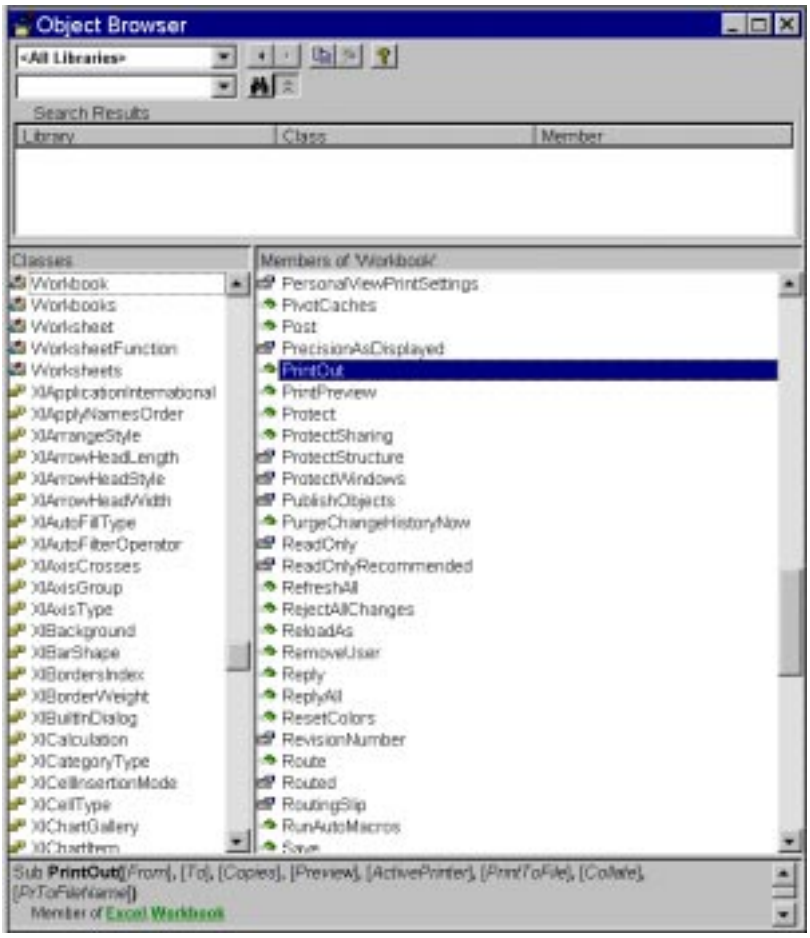

***Figure 6**. Using the Object Browser to determine parameters. When a method is highlighted, the bottom pane shows the calling syntax. Since the Browser gets its information directly from the server, it can't be wrong.*

What does the Browser browse?

For Figure 6, we commented that the Object Browser had been opened from inside Excel. That wasn't really necessary. You can use the Object Browser from any of the Office tools to open and explore any registered type library. You can even look at the objects from multiple type libraries at the same time.

To open a type library so the Object Browser can display its contents, choose Tools|References in the Visual Basic Editor. The dialog shown in **Figure 9** is displayed. Check the libraries you want to add to the Object Browser, and then choose OK. (Be aware that, if you're actually writing code in the VBE, referencing type libraries in this way has consequences for your projects. So be careful what you actually save.)

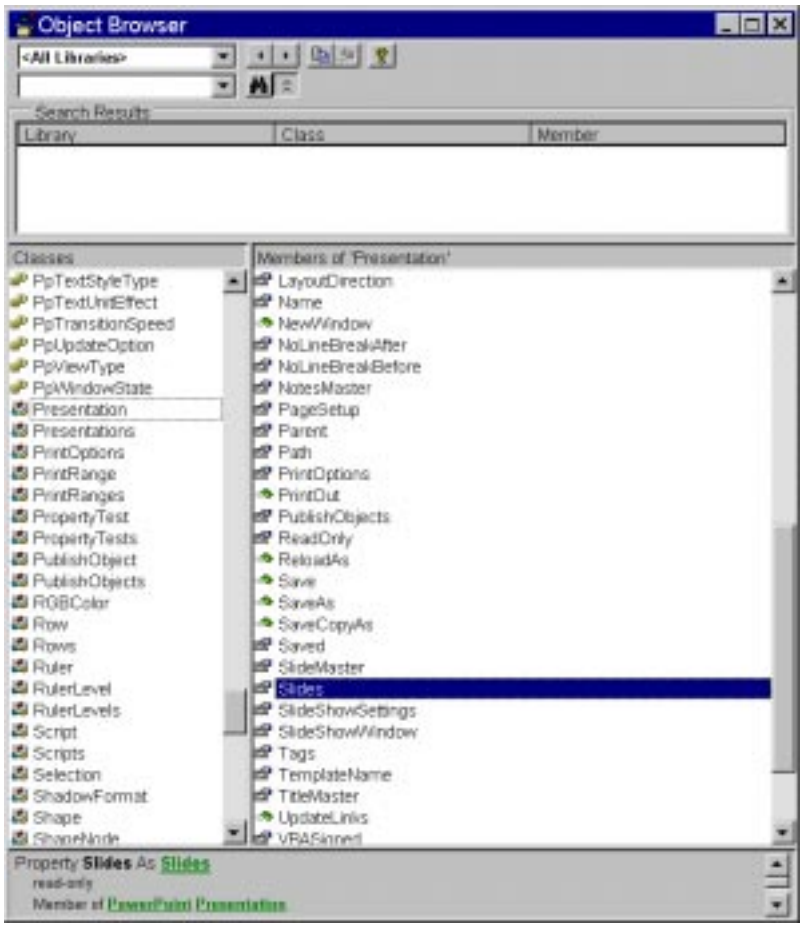

***Figure 7**. Exploring the object model. When an item is underlined in the bottom pane, you can click on it and change your focus in the Browser. Click on Slides, underlined here, to change to the display shown in Figure 8.*

Within the Object Browser, you determine whether you see information from one type library or all of them with the drop-down list in the top left-hand corner. There's no way to choose a varied subset of the open libraries, however—your choice is all or one.

At your command

The Visual FoxPro Command Window is another powerful tool for learning about Automation servers. Once you've read what Help has to say and looked it up in the Object Browser, sometimes you just need to try it. That's where the Command Window comes in.

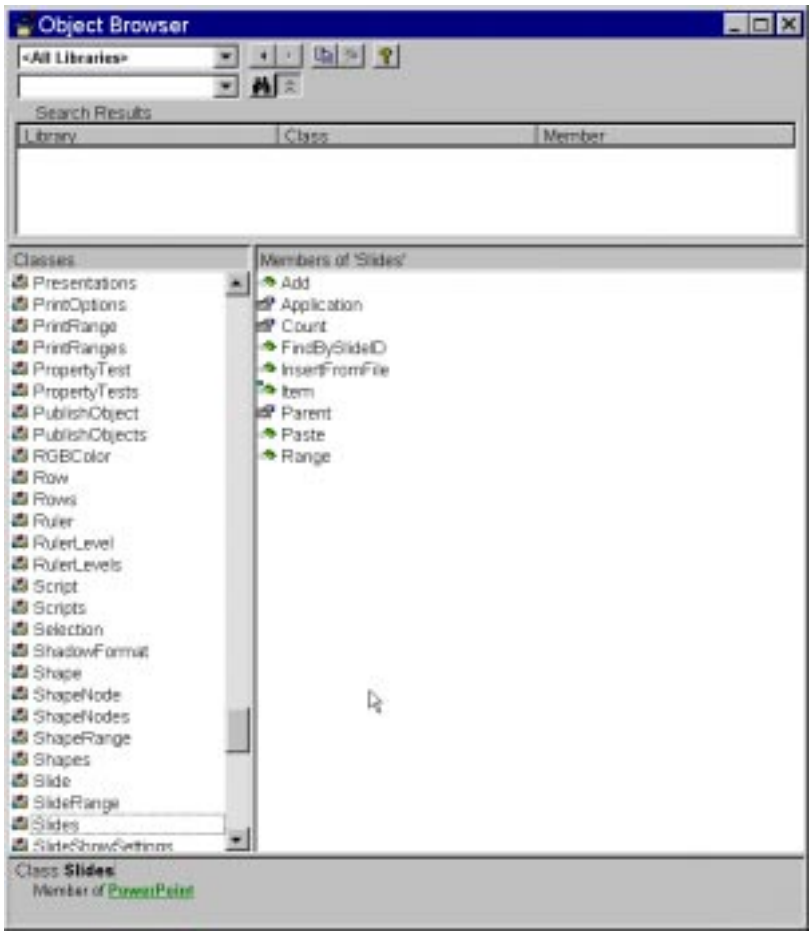

***Figure 8**. The Slides collection. Clicking on the reference to the Slides collection shown in Figure 7 produces this display in the Object Browser. The Browser makes it easy to explore the relationships among objects in the hierarchy.*

Just as it does in every other aspect of working in VFP, the Command Window lets you try things and see what happens without the overhead of building entire applications or setting up complex scenarios. Create a reference to the appropriate server and try the sequence of commands one by one, observing the results as you go.

You can query the value of a property with ? (assuming the value is printable) or execute a method. Even the VFP Debugger can be used in a limited way. The limit is that properties of COM objects are visible in the Debugger only after they've been accessed from VFP. You can't just drill down into COM objects in the Debugger the way you can into VFP's own objects. Too bad.

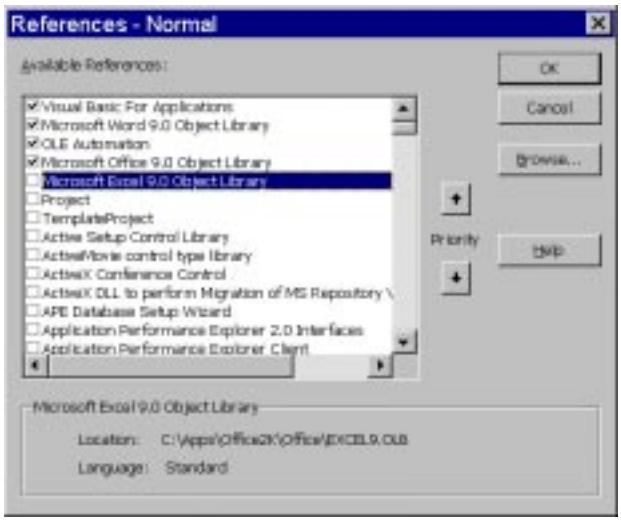

***Figure 9**. Adding type libraries. You can examine all kinds of type libraries using Office's Object Browser.*

We've found it very useful to keep the Command Window visible as we write code. Some commands, particularly those with lots of specified named parameters (and lots more omitted parameters), can be particularly gruesome to get right. Try them in the Command Window, and when they're finally correct, cut and paste them into your code. One big drawback: the Command Window doesn't do #DEFINEs. Either set a variable in the Command Window, or use the corresponding values (remembering to change them to #DEFINEd constants when you paste into your code).

You may find that you get a lot more out of this book if you "work along" in the code. You'll see that we've provided the values for each of the constants in #DEFINE statements at the top of the code. While a #DEFINE line by itself does you no good in the Command Window, cutting and pasting an entire section from the HTML Help version of this book into the Command Window, and then right-clicking and selecting Execute runs the code correctly.

Another really cool piece is that you can move back and forth between doing things interactively and doing them with Automation. That is, when you have an instance of a server available from the Command Window and visible, you can switch over to the server application and work with it interactively, then come back to VFP and check the values of properties that were just set by your action, or execute a method, or set some other properties.

While trying to understand how a particular feature works, we often try something from the Command Window, then switch over to the server application to see the result, then hit Ctrl-Z (Undo) in the server to reverse that action before we go back to VFP and try a different parameter or value or approach. Perhaps more than any other, this ability drives home the reality that Automation really is just one more way to do the same things a user can do interactively.

We encourage you to explore the servers in the Command Window. It really helps to see instantly just what that line of code does (or it becomes an instant approach to finding a syntax error, which is still very helpful, but not nearly as much fun to watch).

On-line and print resources

There are a number of references available for the Office servers besides their respective Help files. Microsoft Press offers a *Visual Basic Programmer's Guide* for both Office 2000 and Office 97. Each is available both in book form and on-line. Since Microsoft is in the habit of rearranging its web site regularly, the best way to find the on-line versions is to search microsoft.com for "Visual Basic Programmer's Guide."

The VBA Help files are also available in printed form. If you'd rather work with a paper copy, you can order them from Microsoft Press. Look for the *Office Language Reference* (or the *Language Reference* for the individual application you're interested in). The *Language Reference* guides are available on the Microsoft web site, too, in case you find yourself stuck somewhere without the Help file.

Microsoft's web site for Office development is located at msdn.microsoft.com/officedev/ (at least it was at the time of this writing). Check it out for official support, technical articles, bug fixes, and so forth, as well as pointers to other useful sites.

Once you become comfortable enough reading VBA code, the various Office and Visual Basic magazines and journals can be useful resources. Take a look at *Microsoft Office & Visual Basic for Applications Developers* (www.OfficeVBA.com) and *Woody's Office Watch* (www.woodyswatch.com) for starters. Also, a new journal from Advisor Media, *Advisor Expert: Microsoft Outlook and Exchange*, looks like it will cover Outlook automation (www.Advisor.com). The major FoxPro magazines, *FoxPro Advisor* (www.Advisor.com) and *FoxTalk* (www.pinpub.com), cover Automation occasionally—the Office servers are the Automation target only for some of those articles.

For immediate help when you're stuck, your best bet can be to try the FoxPro forums and newsgroups. The FoxPro community has a well-earned reputation for its helpfulness—enough members are doing Automation work that simple questions are answered quickly. More difficult ones sometimes go unanswered. There are on-line communities for the Office applications, as well. We have less experience with them, but we have found those we've dealt with to be friendly and knowledgeable. See Appendix A, "On-line User Communities," for a list of on-line user communities for Visual FoxPro and the Office applications.

Taking up a collection

The object models of the Office applications (along with most COM servers) contain lots of collections, the OOP world's version of arrays. A collection contains references to a number of objects, generally of the same type, and provides access to those objects. Generally, a collection has a plural name, such as Slides, and contains objects that are referred to in the singular: the Slides collection contains a series of Slide objects. Just to be sure that you're good and confused while looking in the Help file, most of the time, you access the collection by a property of the same name: the Presentation object has a Slides property, which references the Slides collection. The Slides collection references a series of Slide objects. Just remember that the collection is plural (as is usually the property to access it), and the object is singular, and you shouldn't have much trouble.

Most collections, including those in Office, have a few standard methods and properties. The Count property tells you how many items are in the collection. Item, which is a method in some collections and a property in others (it's a method in the Office apps), provides access to the individual members of the collection.

Item typically takes a single parameter or index (number) and returns a reference to that member of the collection. In many cases, you can specify the item you want by name rather than number. For example, consider Visual FoxPro's Projects collection (which is a COM collection, rather than native to VFP). If the TasTrade project is open and is the first open project, you can access it as _VFP.Projects.Item[1] or _VFP.Projects.Item["TasTrade.PJX"].

In addition, most collections let you omit the Item keyword and simply attach the parameter/index to the collection name. So you can write _VFP.Projects["TasTrade.PJX"] or _VFP.Projects[1] and still get a reference to the project.

As with Visual FoxPro's arrays, you can use either square brackets or parentheses to access the elements of a collection. In this book, we use square brackets for both arrays and collections and leave parentheses to indicate functions and methods.

Changing the collection

Once you get past Count and Item, there's more diversity. Most, but not all, collections have an Add method, which allows you to add a new item of the appropriate type to the collection. For example, you use the Add method of Word's Documents collection to create a new, empty document, and the Add method of Excel's Workbooks collection to create a new, empty workbook. In fact, you also use the Add method of Excel's Worksheets collection to add a new worksheet to an existing Workbook.

There's no common technique for removing items from collections. That's because some types of items remove themselves when they no longer belong in the collection. For example, when you close a Workbook in Excel, it removes itself from the Workbooks collection.

The methods that do remove objects from collections tend to belong to the object itself, not to the collections. Although adding and removing may seem like complementary operations to us, from an object point of view, they really aren't. When you're adding, you have only the collection to work with; you don't yet have the thing you're adding. When you're ready to remove it, you have it in hand and can ask it to remove itself. So Add methods belong to collections, while Remove methods (or the methods that cause members to be removed) belong to members of a collection.

Some collections have a fixed number of entries. For example, the Borders collection in Excel represents the four borders of a range. This collection always has four items; there's no way to add items to it or remove items from it. (Word and PowerPoint also have similar Borders collections, with a fixed number of entries.)

Other collections are modified by other actions in the system. For example, Word's Revisions collection contains the changes that have been made to a document. You can't add or remove items directly because they're handled through a different mechanism.

Self-referential object models

When you start digging around the Office object models, it can be a little confusing (okay, very confusing!). You find that one object has a property that's a reference to a collection of another sort of object, or maybe it's a reference to another object. The property frequently (but not always) has the same name as the collection or other object. When you're digging in Help, it sometimes takes two or three jumps to get to something useful.

Here's an example. Say you start on Word's object model diagram and click on the Documents collection. That takes you to the Help page for the Documents collection (shown in Figure 2—look behind the dialog). What you're probably interested in, though, is the Document object, not the collection, so on the Documents page, you click on Document. That brings you to the Help page for Document. From there, you may decide to find out about the Paragraphs property, so you click Properties. **Figure 10** shows the Document page and the resulting dialog.

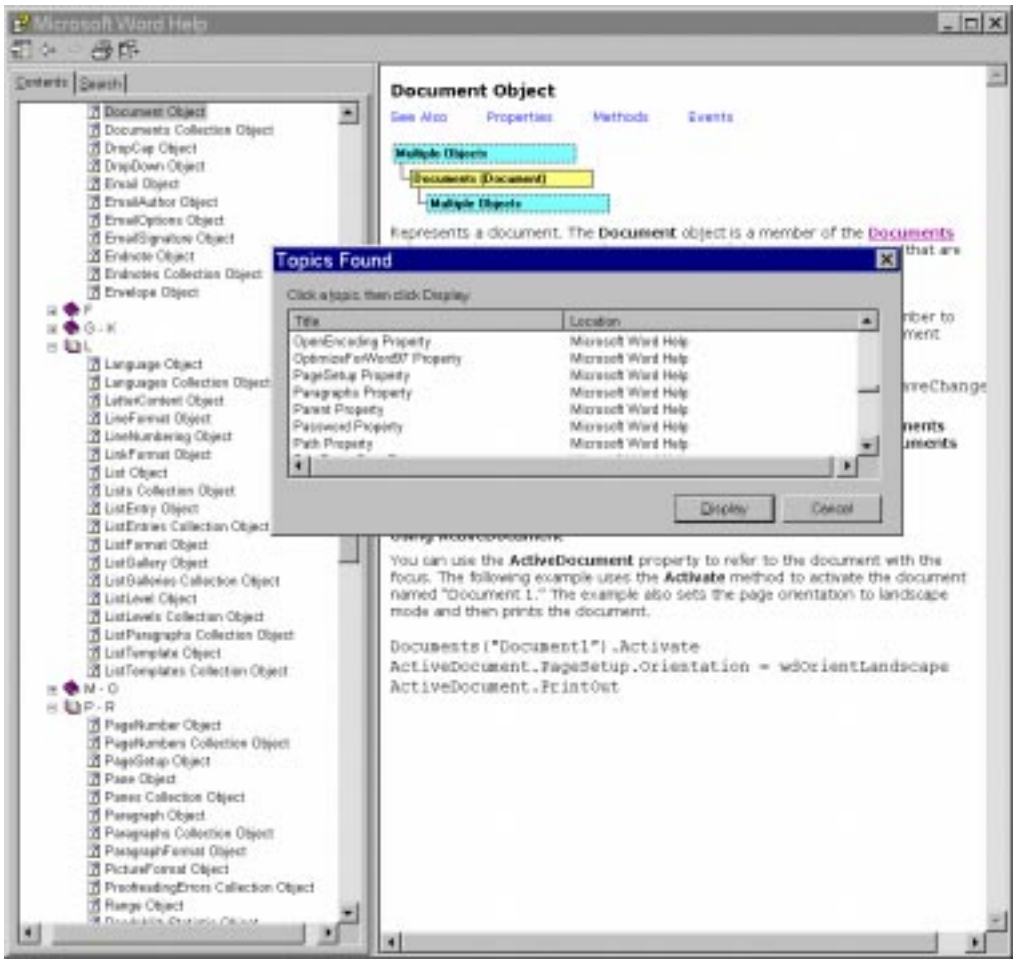

***Figure 10**. From Document to Paragraphs. Many objects have properties that reference collections, using the same name as the collection. While this makes sense, it can make getting help a little long-winded, especially since often, we're ultimately headed for the individual property, not the collection.*

Choosing the Paragraphs property from the dialog brings up Help for that property, as shown in **Figure 11**. To get to Help for the Paragraphs collection requires a click on Paragraphs; to get to the actual Paragraph object—usually our actual destination—requires yet another click once we get to Paragraphs. (The See Also for the Paragraphs property does actually offer a direct jump to Paragraph, but to find that out, you'd have to click on See Also.) This is one of our least favorite things about the VBA Help system. We wish it were smart enough to offer some kind of consolidated help for the property and collection.

***Figure 11**. Not so helpful. This Help entry and others like it, while accurate, add an extra step when you're navigating from one object to another in the Help system.*

There's a flip side to the issue of same-named properties and collections/objects. In some situations, the properties that access the collections or other objects don't have the same names as their targets. For example, a number of objects reference Word's ParagraphFormat object through a property named Format. Since the reference properties almost always use the object names, watch out for the special cases.

Moving on

One of the biggest lessons we've learned in writing Automation code is that you can't automate what you don't understand. In this chapter, we've explored different ways to become familiar with the Automation servers in Microsoft Office, but there's one more that we haven't mentioned.

Use the products. We use Office. Word, in particular, is part of our normal working environment, and much of what we know about automating it comes from hard-won expertise in using it. (On the other hand, we'd be lying if we didn't admit that we've become more powerful Word users through things we learned from Automation programming.)

When you're stuck on a hairy Automation problem, go do it interactively a few times until you really understand both the process and the result. You wouldn't expect to be able to write good programs in a language in whose environment you weren't comfortable working. Why would you expect to do it with Automation?

Add that kind of practice to the exploration tools described in this chapter, and you'll be comfortable in the Office Automation environment before you know it. Now, on to Visual FoxPro's role as Automation client.