Program Word with VBA 5.0

Click & Retrieve Source CODE!

Learn to manipulate a Word document by writing a VBA procedure.

by Chris Barlow

If you read this column regularly, you know that it goes where the VBA action is. It started with VBA in Excel 5.0 and then moved up to VBA in Visual Basic 4.0. Now, with the release of Office97, Microsoft has completed its long-term plan of making Office programmable by embedding VBA in all the Office applications. You can write a routine in VBA to control Excel, Project, Word, Outlook, Access, and so on. No more struggling to learn a different macro language for each different application. No more translating standard routines from one language to another. You can command the full power of Visual Basic and use Office97's rich feature set to write your own procedures to extend the functionality of these applications.

Even better, Microsoft is licensing VBA so that other vendors can embed the Visual Basic programming language in their applications. Soon you should see VBA as an integrated part of CAD programs, drawing programs, and desktop publishing systems. Now is the time to get started. VBA5 will open up significant opportunities for anyone proficient in Visual Basic.

VBA5 is an enhancement to the VBA language that was first released in Excel 5.0 and Project 6.0, then upgraded in Excel 95 and Access 95. Because VBA is written largely in C++ rather than Intel-specific assembler code, it is portable to many platforms. Microsoft is releasing VBA5 for all 32-bit Windows 95 and Windows NT platforms. It will also be available on the Macintosh/PowerPC. There is even some talk of a Unix version.

VBA5 functions through "host applications" such as Excel or Word, which provide the facility to save the VBA source code in modules and share the memory space between the application and VBA. Therefore, these host applications must be running in order to run VBA code. Each host application can provide a different implementation of Visual Basic. For example, while Word 97 stores VBA code in documents or templates, Access 97 provides a slightly different implementation and stores code in the database and uses a different forms engine.

VBA is part of a scalable family of products. The lowest end, VBScript, is a subset of the VBA language designed for use in browsers. At the high end, the standalone product, Visual Basic 5.0, also will use the VBA5 language engine and IDE. But, among other things, VB5 will extend VBA to allow the creation of standalone EXE files and ActiveX documents and controls.

LEARN VBA FOR WORD

Before you can write a VBA procedure to manipulate a Word document, you need to learn a bit about Word's object model. When you write a VBA procedure, you interact with Word through its exposed OLE properties and methods. Although the model is large, with hundreds of properties and methods, you only use a small number of these objects in most of your VBA procedures. If you use the macro recorder, the debugger, and the object browser, you find that Visual Basic helps you learn about these objects. Yes, Word now has a "macro recorder" that records your keystrokes and translates them into VBA code. Using the macro-reader is the ideal way to build up your own library of useful procedures and learn Word's object model.

The Word object model is logical and easy to understand. Its objects range from Document objects at one end of the spectrum to Character objects at the other. When Word is running, its highest level is the Application object. This object has a collection of Document objects that refer to each of the open documents. As with any collection, you can refer to an element of the collection by an ordinal index number. For example, Documents(1) refers to the first Document object in the Documents collection. You can refer to the document you are editing in Word—the active document—by using the ActiveDocument prop-

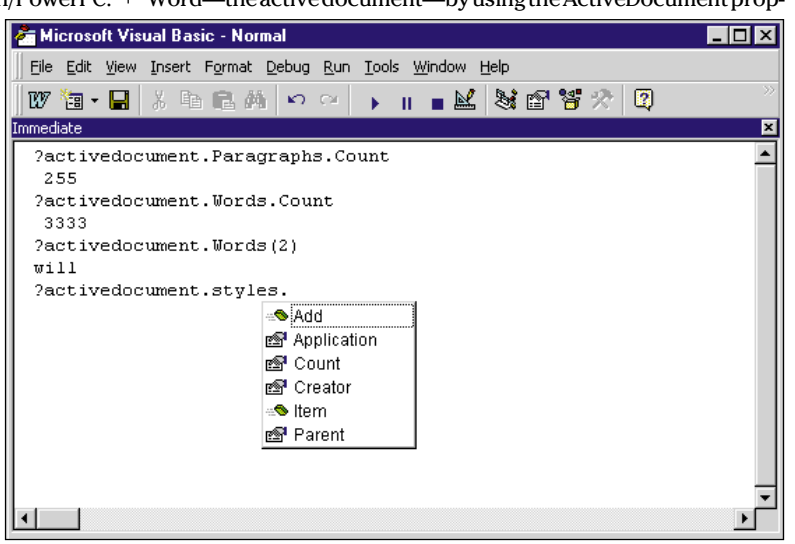

FIGURE 1 ***Immediate Window.*** *Notice how the same popup features available in the editor carry over to the Immediate Window.*

Chris Barlow is president and CEO of SunOpTech, a developer of manufacturing decision-support applications, including the ObjectBank and the ObjectJob Systems, where he and Ken Henderson hold U.S. Patent #5,550,976 related to software for decentralized distributed asynchronous object-oriented systems. Chris holds degrees from Harvard Business School and Dartmouth College, where he worked with Drs. Kemeny and Kurtz on the Basic language. Reach Chris on the Internet at ChrisB@SunOpTech.com or through SunOpTech's World Wide Web server at www.SunOpTech.com.

erty of the Application object.

Each Document object has collections of Paragraph objects, Word objects, and Character objects. As with any collection, you can refer to an item of the collection by an index. Therefore, ActiveDocument.Words(15) returns the 15th word in the document. Similarly, ActiveDocument.Characters(202) returns the 202nd character in the document. You can also combine these objects together so that ActiveDocument.Words(5).Characters(2) returns the second character in the fifth word in the active document.

You can begin to work with Word's object model by using the Immediate Window that is part of the VBA IDE. The Immediate Window lets you enter VBA statements and see the results. Try this out now by starting Word and loading a document. Then select the Macro menu item from the Tools menu and click on Visual Basic Editor (or press Alt-F11) to display the VB editor. The Visual Basic editor pops up in a separate window (although it runs in Word's memory space). Within the VB editor, select the Immediate Window menu item from the View menu (or press Ctrl-G) to display the Immediate Window where you can type in VBA code.

Try some of the examples you see in Figure 1. Notice the feature called List Properties/Methods. As you type, this feature enables the editor to pop up a list box containing the properties and methods of the object. You'll find this to be a huge help while learning the object model. You can use the For Each statement to iterate through all the items in a collection. This statement assigns the next item in the collection to the variable each time through the loop. You don't need to know the index of the first and last items in the collection. Before this statement was available in VBA, you needed to get the Count property of a collection to establish the last item, and you needed to know whether the collection was one-based or zero-based to establish the first item. Try typing these statements in the Immediate Window and see how they operate:

```
For Each x In ActiveDocument.Styles: _
   ?x: Next
For Each x In Documents: ?x: Next
For Each x In ActiveDocument._
   Paragraphs: ?x.Style: Next
```

The first statement displays all the styles in the active document by using the Styles collection and printing the default property of the Style object—its name. The second statement steps through the Documents collection and displays the document name. The last statement steps through the Paragraphs collection of the active document and displays the name of the Style associated with that paragraph.

LEARN BY RECORDING

When you sit down to write your first VBA procedure for Word, you may not know where to start. How do you insert text into Word? How do you move through the document? Let the macro recorder help.

For example, suppose you are typing a letter. You aren't sure of the person's address, so for now you just type his or her last name. After you finish the letter you find the last name, look up the proper address from your contact manager, and type the address into the letter. How would you write a procedure to automate part of this process?

Begin by starting a new Word 97 document and typing the text of the letter with only the last name where you want the full name and address. Then turn on the macro recorder so it begins recording your keystrokes, translating them into VBA, and saving them for you to review, edit, or run later. The easiest way to turn on the macro recorder is to click in the panel labeled REC on the Status Bar at the bottom of the Word screen. The Record Macro dialog that appears lets you enter a name and description for the macro, assign it to a toolbar or keyboard shortcut, and decide whether to store it in a template or the open document. Enter ReplaceName for the macro name, and click on the OK button.

You will see that the image of a small tape recorder appears next to your mouse pointer. You will also see a small toolbar with Stop and Pause buttons appear. Now click on the Find menu item on the Edit menu to start the Find dialog. Enter the last name that you typed in the letter, and click on the Find Next button to find and select the last name. Then click on the Cancel button to

```
Attribute VB_Name = "NewMacros"
Option Explicit

Sub ReplaceName()
Attribute ReplaceName.VB_Description = _
  "Macro recorded Monday, September 16, 1996 by " & _
  "Chris Barlow"
Attribute ReplaceName.VB_ProcData.VB_Invoke_Func = _
  "Project.NewMacros.ReplaceName"
Dim txt$
txt = InputBox("Enter LastName")
Selection.Find.Execute (txt)
txt = InputBox("Enter Full Name")
txt = txt & vbLf & InputBox("Enter Organization")
txt = txt & vbLf & InputBox("Enter Street")
txt = txt & vbLf & InputBox("Enter City, State Zip")
Selection.TypeText Text:=txt
Selection.TypeParagraph
End Sub

Sub ReplaceNameDB()
Dim txt$
txt = InputBox("Enter LastName")
Selection.Find.Execute (txt)
Selection.TypeText Text:=LookupName(txt)
Selection.TypeParagraph
End Sub

Function LookupName(Lastname$) As String
Dim ws As Workspace
Dim db As Database
Dim rs As Recordset
Dim SQL$

SQL = "Select Contact, Name, Addr1, City, State, Zip"
SQL = SQL & " From Contact Where Last_Name = '"
SQL = SQL & Lastname & "'"
Set ws = DBEngine.Workspaces(0)
Set db = ws.OpenDatabase("c:\actwin2\database", _
  True, True, "dBase IV")
Set rs = db.OpenRecordset(SQL)
If Not rs.EOF Then
  LookupName = rs(0) & vbLf & rs(1) & vbLf & _
     rs(2) & vbLf & rs(3) & ", " & _
     rs(4) & " " & rs(5)
End If
rs.Close
db.Close
Set rs = Nothing
Set db = Nothing
End Function
```

LISTING 1 ***ReplaceName Document.*** *This source is embedded in the Word document file. Notice the VBA5 Attribute within the module with the procedure description.*

close the dialog. Now type the correct address. Remember to use the Shift-Enter key combination at the end of each address line so that all lines are part of the same paragraph for formatting purposes. Then press the Stop button to stop recording the macro, and press Alt-F11 to jump to the Visual Basic editor. Take a look at the code for the procedure Sub ReplaceName(). You should see something like this:

```
Sub ReplaceName()
Selection.Find.Execute
Selection.TypeText Text:="Mr. Christopher R. Barlow" & _
   Chr(11) & "SunOpTech" & Chr(11) & _
   "1500 West University Parkway" & Chr(11) & _
   "Sarasota, FL  34243-2290"
Selection.TypeParagraph
End Sub
```

You can see from using the recorder that the Find object has an Execute method and that the TypeText method of the Selection object inserts text into the document. The TypeText method replaces any text that you select with the text passed in the Text parameter. Also notice that the Shift-Enter key combination translates to the ASCII linefeed character.

TAKE ADVANTAGE OF THE MACRO RECORDER

This code needs just a few changes to generalize it for future use. For the easiest way to write VBA code, let the macro recorder do the work, and then make a few changes to make the procedure useful in the future.

First, use the object browser to find out how to make the Find method more generic. While you are in the code window, position your cursor on the Execute method and press Shift-F2 to pop up the object browser. Notice how the object browser jumps to the Execute method and displays the possible parameters. The object browser is powerful. It not only shows all your procedures, but it shows the function parameters as well. You can see both built-in and custom properties of all the referenced objects. The object browser form is modeless, and you can search across all or selected type libraries. You can even click on the underlined ListBox in the parameter block for a procedure and jump directly to the description of that object.

There's an even easier way to view parameters. Try this: position your cursor after the word Execute and type a space. A small "tip" window appears with the parameters for the Execute method. The FindText parameter looks like the one to use. VBA includes most of the statements you are familiar with using in Visual Basic 3.0 and 4.0. You can use the familiar InputBox statement to allow the user to enter the last name to find and also to enter all the lines of the address. Change the Text parameter of the TypeText method to remove the hard-coded address; instead, use the txt variable. Notice how the special VBA constant, vbLF, is used for the ASCII linefeed character:

```
Sub ReplaceName()
Dim txt$
txt = InputBox("Enter lastname.")
Selection.Find.Execute (txt)
txt = InputBox("Enter Full Name")
txt = txt & vbLf & InputBox("Enter Organization")
txt = txt & vbLf & InputBox("Enter Street")
txt = txt & vbLf & InputBox("Enter City, State Zip")
Selection.TypeText Text:=txt
Selection.TypeParagraph
End Sub
```

It's as simple as that. You can assign this macro to a keyboard shortcut by selecting the Customize menu item from the Tools menu to display the Customize dialog and clicking on the Keyboard button to display the Customize Keyboard dialog. Then scroll down through the Categories to Macros, select the ReplaceName macro, and press the shortcut you want to use.

DATABASE ACCESS

Any Visual Basic programmer will tell you that one of the most powerful features of Visual Basic is its easy database access. And anyone who works with WordBasic will tell you that easy database access is one of the most sorely missed features. Now you have the full power of Visual Basic database access at your fingertips.

Take your ReplaceName procedure and modify it to look up the name in your Contact database and insert it into the Word document. First, select the References menu item from the Tools menu to display the References dialog. As you write VBA code, Visual Basic looks in the object libraries that are included in this References dialog to validate and compile your code. If you do not include the proper object libraries, you get errors in your code. For example, if you try to reference the Database or Recordset object without including one of the DAO object libraries, VBA will seem to not understand database access. By checking the Microsoft DAO 3.5 Object Library item in the list box, however, you have full access to these objects.

I am in the process of moving from my old contact management software to Microsoft Outlook. This neat new part of Office, also accessible through VBA, will be the topic of several future columns. For now, I'll show you how to write the code to get the contact information from the dBase files used by Symantec's Act! Contact Manager. You should be able to easily modify these procedures to use any database that the Microsoft Jet engine can access.

From Word, press Alt-F11 to jump to the VBA code window, and create a new function called LookupName. When a last name is passed as a parameter, this function looks up the name in the Act! Contact database and returns the full name and address as a string with embedded linefeed characters. Begin by dimensioning local variables for the Workspace, Database, Recordset, and SQL statement:

```
Function LookupName(Lastname$) As String
```

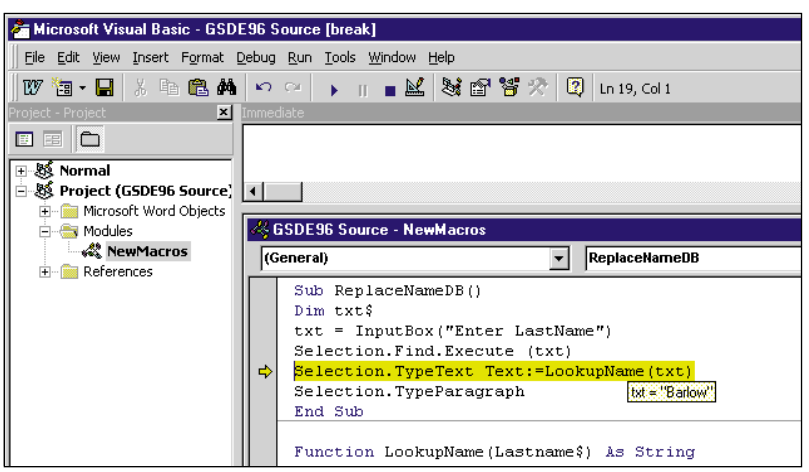

FIGURE 2 ***Debugger.*** *Unlike the VBA2 debugger, the new version lets you change the code or set the next line to execute and continue running.*

```
Dim ws As Workspace
Dim db As Database
Dim rs As Recordset
Dim SQL$
```

Then create the SQL statement to select the proper fields from the Contact table in Act!'s database folder. Notice how I've included the Lastname parameter in the SQL string by separating its value with single quote marks. Many beginning programmers enter this SQL string incorrectly and end up with a SQL statement that has the word "Lastname" rather than the value of the variable Lastname:

```
SQL = "Select Contact, Name, Addr1, City, State, Zip"
SQL = SQL & " From Contact Where Last_Name = '"
SQL = SQL & Lastname & "'"
```

Then open the Workspace, Database, and Recordset with this SQL statement:

```
Set ws = DBEngine.Workspaces(0)
Set db = ws.OpenDatabase_
   ("c:\actwin2\database", True, True, "dBase IV")
Set rs = db.OpenRecordset(SQL)
```

If some records were returned in the Recordset, concatenate the Fields from the Recordset into a single string and set the string to the function name to return that value:

```
 If Not rs.EOF Then
   LookupName = rs(0) & vbLf & rs(1) & _
      vbLf & rs(2) & vbLf & rs(3) & _
      ", " & rs(4) & " " & rs(5)
 End If
```

Finally, close the local objects and set them to Nothing to free their allocated memory:

```
rs.Close
db.Close
Set rs = Nothing
Set db = Nothing
End Function
```

Now copy the ReplaceName procedure that you wrote, and name it ReplaceNameDB. Change the Text parameter of the TypeText method to remove the txt variable, and instead call your LookupName function with the txt variable as the parameter:

```
Sub ReplaceNameDB()
Dim txt$
txt = InputBox("Enter LastName")
Selection.Find.Execute (txt)
Selection.TypeText Text:=LookupName(txt)
Selection.TypeParagraph
End Sub
```

Let's give this a try. Type a sample letter with only the last name where the full name and address should go. Then press Alt-F8, select the ReplaceNameDB macro, and click on the Step Into button so that you can single-step through the code with VBA's integrated debugger. The input box should appear, and you can type the last name and press Enter. Notice that when you hover the mouse over a variable, a tip appears with the value—I love this feature! It makes it easy to watch the flow of the procedure without having to pop up Watch windows (see Figure 2). You can assign this macro to a keyboard shortcut by selecting the Customize menu item from the Tools menu and clicking on the Keyboard button to display the Customize Keyboard dialog. Then scroll down through the Categories to Macros, select the ReplaceNameDB macro, and press the shortcut that you want to use.

I've included complete code for the module (see Listing 1). Obviously, this function still needs some work. For example, what if more than one match exists for the last name? This procedure returns only the first match. You can see how easy it is to add the functionality you need for your ReplaceNameDB procedure. Why don't you try to use this base code to add some new features and e-mail me your best results? You can access the code to my more comprehensive application from the Premier Club of the Development Exchange (for details, see the Code Online box). ×

Code Online

You can find all the code published in this issue of **VBPJ** *on The Development Exchange (DevX) at http://www.windx.com. All the listings and associated files essential to the articles are available for free to Registered members of DevX, in one ZIP file. This ZIP file is also posted in the Magazine Library of the* **VBPJ** *Forum on CompuServe. DevX Premier Club members ($20 for six months) can get each article's listings in a separate file, as well as additional code and utilities for selected articles, plus archives of all code ever published in* **VBPJ** *and* **Microsoft Interactive Developer** *magazines.*

Program Word with VBA 5.0

Locator+ Codes

Listings ZIP file (free Registered Level): VBPJ1296

✪ *Listings for this article plus a Word 97 document that includes the extended source to handle multiple last names (subscriber Premier Level): GS1296P*