Creating an Employee Newsletter Using VisiBase and OLE Automation

by Richard Mansfield

VisiBase, a Visual Database Using OLE Automation

Most significant applications include a scripting (macro) language for automating common tasks. OLE Automation extends this capability to allow you to automate tasks across two or more applications.

VisiBase is designed to show how one Windows-based program controls another Windows program using OLE Automation. This control includes the capability of fully using the features of the second, controlled program. VisiBase demonstrates how to combine the facilities of Visual Basic — notably its built-in Access database engine — with the DTP facilities in Word for Windows.

Telling VisiBase how to talk to Word is simple. Create a macro in Word, then copy the text of that macro and paste it into a Visual Basic 3.0 program. In some cases, you need to change the syntax of WordBasic arguments to make them work correctly with OLE Automation, but this translation is simple, consistent, and easy to remember.

VisiBase is a Visual Basic 3.0 program that uses Word for Windows 6.0 facilities to load a disk file, update a Date Field, locate bookmarks, delete and insert text, insert photographs, and save the modified file back to disk.

With a Visual Basic front end, you can modify data about each employee in a company, including viewing a scanned photograph of the employee and a brief biography. This data, along with the photo, is automatically inserted into a Word template, ready for publication in a company newsletter. The same VB front end could be used, with minor modifications, to automate the creation of security ID badges, a yearbook, and so on.

Project Description and Implementation

VisiBase allows the user to enter or modify the name, date started, .BMP file, and narrative biographical data about each employee in a company. The user can access each record (for editing, deleting, or adding records) by scrolling with the standard scrollbar data control, or by clicking the employee's name within a list box.

When a particular employee is chosen to be the "Employee-of-the-month" (his or her record is the current, visible record) — pressing a Command Button labeled "Change Newsletter" automatically inserts the name, bio, years employed and photograph into a Word template. This template, called EMPLOYEE.DOT, is a page of the company newsletter, a regular feature called "Employee-of-the-Month."

OLE Automation performs the following tasks in VisiBase:

1. It locates the date field in the Word template and updates it to the current date.
2. It locates a headline bookmark and inserts the employee's first and last names. Because the headline should remain on a single line, VisiBase checks the length of the employee's name and adjusts the size of the typeface in Word as necessary.
3. It inserts the employee's photograph.
4. It locates a caption bookmark beneath the photo and inserts a new caption. Visual Basic's Date/Time facilities are used to subtract the employee's start date from the current date. Then the customized caption "Bob Jones has worked for BeBe Incorporated for 3 years" is automatically inserted.
5. Finally, the body of the article is inserted, based on the biographical data in the employee's record.
6. This updated template ("EMPLOYEE.DOT") is then saved to disk and control is returned to the user and VisiBase.

If you wish to fine-tune the EMPLOYEE.DOT template, you can double-click a linked OLE2 VB Control and bring up Word with the template page ready for pre-publication polishing.

Using OLE Automation

Using OLE Automation is about as easy as using application-specific macro languages, such as WordBasic. In fact, most OLE Automation Word is essentially WordBasic with the name of an OLE "object" prepended.

For example, to manually move to the start of a Word document, you press CTRL+HOME. If you record this action in a macro, then open the macro for editing, you will see the following macro command:

```
Sub MAIN
StartOfDocument
End Sub
```

If you were to move the cursor mouse pointer to the top of the Word document from VisiBase via OLE Automation, you simply add the name of the object to the WordBasic command, as in this example:

```
WordObj.StartOfDocument
```

Therefore, where StartOfDocument is a WordBasic command, WordObj.StartOfDocument is an OLE Automation command that does the same thing — it taps into Word and causes the cursor to move to the top of the document.

The Meaning of "Object"

Where did we get "WordObj," the name of this object? And what is an "object?"

In this context, an *object* is simply an outside program, or a subset of that program. In Word for Windows, you define an object (Word.Basic) that allows you to manipulate the Word for Windows application itself — taken as a whole. Word also permits you to create objects of a bookmark, a small selection, a sentence, or a paragraph. Each OLE-capable application contains a list of which of its zones or facilities — in other words, its *objects* — are available for outside manipulation. Applications are said to "expose" their OLE objects. Which objects are exposed are described in each application's manual. In some cases you can manipulate such objects as a toolbar, or a range of cells in a spreadsheet.

Word exposes its entire macro language to OLE automation. This allows you to significantly manipulate Word from outside Word. In this way, we can directly affect the behavior and configuration of Word, and process documents using all of Word's features. You can use the outside, controlling program (in the case of VisiBase, we use Visual Basic 3.0) to query the status of something, to change the status of something, or to carry out an action. You can read and write Properties, or trigger Methods.

In OLE automation, one application controls another application. However, the control is analogous to the way applications can manipulate their own internal objects — their features and data. For instance, you can manipulate internal objects in Visual Basic (its Controls) by finding out (reading) a property of a Visual Basic object (such as the Color property a shape object), changing (writing to) that property, or using a Visual Basic method.

In Visual Basic, the qualities of an object are called *properties* (for example,

Color is a property of a list box). The actions you can take to affect the object are called *methods* (the **Clear** method removes all the entries in a list box). Similarly, when you use Visual Basic to manipulate Word and a document loaded into Word, you will be able to read or write to the qualities of the document (such as querying or changing the font size of some text), or take more direct actions, such as inserting a new photograph into the document.

OLE Automation in Action in a Word Document

Here is a line-by-line description of how we used OLE Automation to update a Word for Windows document. It isn't necessary to study WordBasic to learn what commands you should use to cause particular behaviors in Word. Simply create a new macro in Word by choosing Macro from the Tools menu. Start recording the macro; then choose Edit from the Macro menu and copy the commands that Word has created based on your recorded actions.

Figure 1: The VisiBase Application. In VisiBase, the user sees the window in Figure 1. Most of this is fairly straightforward if you've used Visual Basic. There are five text boxes into which the user can enter the name, start date, filename of the photograph, and a biographical narrative about each employee. In addition, there is a scroll bar with which the user can move through the records in this database (this scroll bar is actually a data control). There is also a list box with employee names that can be clicked as an alternative way to bring up a particular record. Also, Visual Basic uses two files for its database, (NEWSLETT.MDB and NEWSLETT.LDB), which are compatible with Microsoft Access. Therefore, if the user can choose to use Access to open and manage these files.

There are, however, three elements in the VisiBase window which are not typical of traditional databases. First, there is a photo which appears for each employee record. This photo comes on screen as quickly and as naturally as the new text data when the user changes to a different record. Second, there is an OLE Control which displays a typical Employee-of-the-Month page layout for the company newsletter. Third, there is a Command Button labeled "Change Newsletter." This button is the gateway to the power of OLE Automation. Click on this button and Word opens, loads in the Employee-of-the-Month template, and inserts the data and photo from the current record into the template. All this is done automatically without user intervention.

How to Program using OLE Automation

To program OLE Automation, change the information bar to inform the user that OLE Automation is taking place when the Change Newsletter Button Is clicked:

```
Helpbar.Caption = " UPDATING WINWORD DOCUMENT." 'Explain the pause to the user
```

Next, declare an object variable with the **Dim** statement. Declare the arbitrary name WordObj to be a new Visual Basic object:

```
Dim WordObj As object, years As Variant, le As Single
```

(Note that a couple of additional variable declarations are added on that same line, but they are unrelated to this object declaration. They just share the **Dim** line.)

Assign a specific object reference to the new object variable WordObj using the **Set** statement. Use the Visual Basic **CreateObject** function (which defines the object) together with the **Set** statement to bring to life an OLE object. The **CreateObject** function returns an object and the **Set** statement assigns this new object to the previously declared object variable, WordObj:

```
Set WordObj = CreateObject("Word.Basic")
```

There are two arguments you must provide to the **CreateObject** function: the *name* of the application that is exposing the object, and the particular *type* (or "class") of object being accessed. In this case, the application name is Word and the class is Basic.

There is now a "tube" connecting the Visual Basic application to Word for Windows through which data and commands can pass bi-directionally. As we'll see, this object linking allows the Visual Basic application to contact and control Word virtually without restriction. To invoke a WordBasic command, use the object variable to address WordBasic. OLE Automation handles the rest of the details. This direct access to WordBasic is superior to the older DDE approach (sending instructions via the Visual Basic **SendKeys** statement).

To place the text and photo from the current record into the Word template document, load the template into Word. You can just turn on a macro in Word and record the steps that you wish to automate. You can then view the text of the macro to see the list of commands.

First, open the document, move to the start of the document, locate the date field and then force it to update. To capture the correct WordBasic programming to accomplish these steps, start Word. Turn on macro recording: press ALT+T, M, type in "testmac" and then press ALT+O, ENTER. Word records all keystrokes.

Using either the mouse or the keyboard, load the file EMPLOYEE.DOT and press CTRL+HOME to go to the top of the document. Locate the date field by pressing ALT+E, G, ALT+W and select "Field." Now press ALT+E and select "Date" and press ALT+T, ESC. To update the date field, press F9. Stop the macro. Then open the macro and copy the WordBasic commands into your outside OLE Automation routine. Press ALT+T, M and select "testmac," and press ALT+E. You should see the following WordBasic commands:

```
Sub MAIN
FileOpen .Name = "EMPLOYEE.DOT", .ConfirmConversions = 0, .ReadOnly = 0, .AddToMru =
0, .PasswordDoc = "", .PasswordDot = "", .Revert = 0, .WritePasswordDoc = "", .WritePasswordDot = ""
StartOfDocument
EditGoTo .Destination = "d'DATE"'
UpdateFields
```

§

End Sub

How to Translate WordBasic into OLE Automation

There are three things to learn from this WordBasic code:

1. To use with OLE Automation, remove all the WordBasic identifiers, such as ".Name" = and ".ConfirmConversions =".

2. Eliminate any default arguments in the argument list, such as empty text variables "" or items set to 0.

3. Add the object variable name (that we dimensioned earlier in our OLE Automation program fragment: .WordObj) to each line of automation programming.

In WordBasic, you can now mix and match arguments. Since each argument in WordBasic is identified by such tags as .PasswordDot = or .Revert =, WordBasic you can leave some of them out of the argument list if you want to use the defaults. What's more, WordBasic also allows you to list the arguments in any order you want! See the following example in WordBasic:

```
FileOpen .Name = "EMPLOYEE.DOT", .ConfirmConversions = 0
means the same as this:
FileOpen .ConfirmConversions = 0, .Name = "EMPLOYEE.DOT"
```

You cannot mix and match arguments in Visual Basic. Visual Basic still requires traditional argument lists. Position is important and default arguments must be represented (at least as a location signified by a comma) if you want to set any arguments past them further down in the list.

Translating from WordBasic to OLE Automation involves removing the dotted labels in the argument list and removing any null or default arguments that we wish to eliminate. For Visual Basic OLE Automation, translate WordBasic's first line that opens the file into a Visual Basic OLE Automation first line that does the same thing:

```
'WordBasic Code.
FileOpen .Name = "EMPLOYEE.DOT", .ConfirmConversions = 0, .ReadOnly = 0, .AddToMru =
0, .PasswordDoc = "", .PasswordDot = "", .Revert = 0, .WritePasswordDoc = "", .WritePasswordDot = ""

.VBOLEAutomation
WordObj.FileOpen directoryname$ & "EMPLOYEE.DOT"
```

Note that *Directoryname$* is a variable in our Visual Basic program that identifies the directory in which VISIBASE.EXE resides. VISIBASE expects to find the Word document EMPLOYEE. DOT in this directory. To create the complete path, append *directoryname$* to the file name. The true, elemental translation of the WordBasic line is simply WordObj.FileOpen "EMPLOYEE.DOT".)

Note: 1. In the example above, the .Name = label is removed from the WordBasic version, the default arguments are removed from the argument list,

and the object variable for OLE Automation: “WordObj” is added to the command “FileOpen,” separated by a dot:

```
WordObj.FileOpen
```

To complete our translation of the WordBasic macro commands into Visual Basic OLE Automation commands, we follow the same three rules:

```
StartOfDocument
EditGoTo .Destination = "d'DATE"'
UpdateFields
'Visual Basic OLE Automation
WordObj.StartOfDocument
WordObj.EditGoTo "d'DATE"'
WordObj.UpdateFields
```

The rest of our transformation of WordBasic macro code into VB OLE Automation code follows those same three rules. The following is the complete text of the OLE Automation activity that is triggered when you click command button labeled “Change Newsletter”:

```
Dim n As String, a As String
helpbar.Caption = " UPDATING WINWORD DOCUMENT." 'Explain the pause to the user
Dim WordObj As object, years As Variant, le As Single
Set WordObj = CreateObject("Word.Basic")
WordObj.FileOpen directoryname$ & "EMPLOYEE.DOT"
WordObj.StartOfDocument
WordObj.EditGoTo "d'DATE"'  ' Locate Date Field.
WordObj.UpdateFields        ' Update date.
WordObj.EditGoTo "headname" ' Find headline bookmark.
WordObj.EndOfLine 1         ' Select entire line.
        ' Calculate length of Employee Name.
n$ = txtFirstName.Text & " " & txtLastName.Text
le = Len(n$)
Select Case le 'change headline fontsize to avoid line breaks.
   Case 0 To 17
      WordObj.FontSize 36
   Case 18 To 25
      WordObj.FontSize 24
   Case 25 To 200
      WordObj.FontSize 18
End Select
WordObj.Insert UCase$(n$)   ' insert Employee name.
WordObj.EditGoTo "photo"    ' locate "photo" bookmark.
WordObj.CharRight 1, 1      ' select photo.
WordObj.EditClear           ' delete old photo.
           'insert photo
WordObj.InsertPicture directoryname$ & datEmployeeDatabase.Recordset.Fields(3).Value
WordObj.EditGoTo "caption"  'locate "caption" bookmark
' figure out how many years the employee has worked for BeBe
If txtStartDate.Text = "" Then 'if no start date is in the record
   a$ = txtStartDate.Text     'simply use employee name as caption.
   GoTo nodate
End If
           'calculate how many years employed
years = Year(Now) - Year(txtStartDate.Text)
a$ = n$ & " has worked at BeBe for "
Select Case years
   Case Is < 1
```

§

!Unexpected End of Expression§

```
    a$ = a$ & "less than one year."
    Case 1 To 2
    a$ = a$ & "about a year."
    Case Else
    a$ = a$ & years & " years."
End Select
nodate:
WordObj.Insert a$       'insert caption
WordObj.EndOfDocument 1 'select all text to end of document
WordObj.EditClear       'delete it
WordObj.InsertPara      'move down two lines
WordObj.InsertPara
WordObj.Insert txtBiography.Text   'insert biographical narrative
WordObj.FileSave            'save the file to disk
Set WordObj = Nothing 'release memory/resources
helpbar.Caption = ""
End Sub
```

At the very end of this code, use the following command:

```
Set WordObj = Nothing
```

Setting an object variable name to “Nothing” vaporizes that object, freeing any system resources it was consuming. If you leave this line out, the user’s Windows system resources will remain unnecessarily depleted. Repeated calls to **CreateObject** will use system resources until Windows begins to slow down and display poor screen redraws.

Technical Notes

To use this program you must know in advance which version of Word for Windows the user has installed on his or her computer. We must know the location of the EMPLOYEE.DOT file. Because Word 2.0 reveals its directory within WIN.INI and Word 6.0 has a private .INI file for that purpose, we assume that the user has Windows in C:\WINDOWS and is using Word 6.0. If C:\WINDOWS is not the location of the Windows directory, change the following line in the PROCS.BAS module (the **GetWordDir** function):

```
d = GetPrivateProfileString("Microsoft Word", "programdir", "not", answerstring, 49, "c:
\windows\Winword6.ini")
```

Leave this as a single line in the program. Don’t press **Enter** if you must edit the final entry “c:\windows\Winword6.ini” to something like “d:\windows\Winword6.ini.”

If the user is still using Word 2.0, you must replace the above line with the following:

```
d = GetProfileString("Microsoft Word", "programdir", "not", answerstring, 49)
```

Replace the declaration for the API function. GetPrivateProfileString locates a custom .INI file, whereas GetProfileString looks for an entry within WIN.INI. Type the following declaration for GetProfileString into the Declarations section of PROCS.BAS all on a single line:

```
Declare Function GetProfileString% Lib "Kernel" (ByVal lpApplicationName$, ByVal lpKeyName As Any,
ByVal lpDefault$, ByVal lpReturnedString$, ByVal nSize%)
```

MiscFlags

Because OLE is still a young, evolving technology, some system configurations trigger unusual in-place editing. Sometimes an application's toolbars will appear instead of the complete application. Sometimes you can edit the control of the OLE control, but the type is too small to do so. Make sure that the user can reformat the OLE control (the sample Employee-of-the-Month that is displayed within VISIBASE). Ensure that the OLE control MiscFlags property is set to 2. This prevents any attempt to edit in-place. It starts Word and loads the Employee-of-the-Month template into Word.