



# Program Excel 97 with VBA



**VBA5 includes much of the same functionality as VB, making it easier to write code for Excel.**

by Chris Barlow

When Microsoft put VBA in Excel 5.0, the world of Excel programming changed. All the powerful features of Excel were available to the programmer—not through some hard-to-comprehend macro language, but through standard Visual Basic code. Since then, things have only gotten better. With the release of Office 97, you can write a routine in VBA to control Excel, PowerPoint, Word, Project, Outlook, and Access, and use Office's rich feature set to write your own procedures that extend the functionality of these horizontal applications.

If you have been working with VBA in previous versions of Excel, then Excel 97 will be an easy transition. There have been only minor additions and changes to the actual VBA syntax and the Excel object model. If you have not programmed Excel with Visual Basic, you are going to need to learn a bit about Excel's object model. Fortunately, although the object model has hundreds of properties and methods, you will use only a small number of these objects in most of your VBA procedures. The built-in macro recorder, debugger, and object browser will help you learn about these objects. For more information, see my last Getting Started with VBA column, "Program Word with VBA 5.0" [VBPJ December 1996].

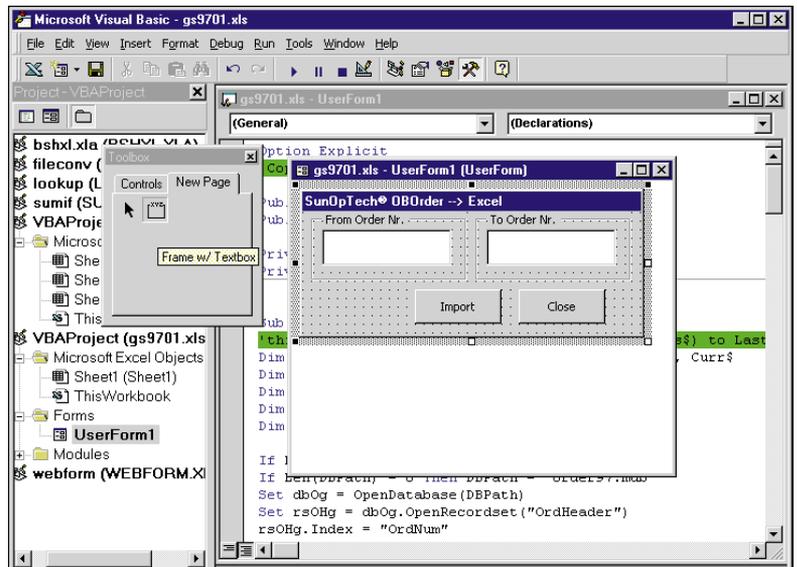
You will see a big difference in the Visual Basic development environment built into Excel. All the Office 97 applications now share the same development environment. This integration makes it even easier to move from developing an extension for Excel to developing one for Word. The best news for those of you who have worked with VBA in Excel 5 and Excel 95 is that the new VBA includes much of the same functionality that has always been in the Visual Basic standalone product. One of the things that always irritated me when programming in VBA was that I could not edit a line of code while running a procedure.

*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.*

The programming environment forced me to stop the procedure, make the code change, then start the procedure from the beginning again. Now this may sound picky to C++ programmers who are used to recompiling and relinking their entire applications to make any code change, but as a Visual Basic programmer, I found this process really disrupted my productivity. Finally, with the new VBA engine, you can stop the execution of a VBA procedure, change a line of code, and continue with the procedure. Thanks, Microsoft!

Start up the Visual Basic development environment from the menu or by pressing the Alt-F11 shortcut key combination. Notice it looks a bit like VB4 with a Project window, Toolbox window, Properties window, and Code window. Unlike VB4, however, you can leave all these windows floating or docked at the edges of the screen. Surf the menu items, and you'll see many of the familiar functions. There are major enhancements to the IDE, though, that will also be in the next version of Visual Basic. The editor now supports drag/drop editing in the manner that Word does—highlight one or more lines of code and drag them where you want them. Another useful feature is the small "tooltip" window that pops up to let you select the proper statement, method, or procedure. Take a look at the Project window. You can save your code within the active workbook or in another workbook. You can also access the properties and events of the worksheet.

One of the best additions to VBA is the Microsoft Forms engine that lets you develop the same kind of forms you are used to in VB4. Prior to this implementation of the new forms engine, you were limited to creating simple dialog boxes. Now you can add any ActiveX control to your form. You can also combine controls and add them to your toolbox. For example, I often put a TextBox



**FIGURE 1** Add Controls to Your Toolbox. You can add several controls to your toolbox as a single control, so you can easily add this new grouped control to any form.



## GETTING STARTED WITH VBA

control inside a Frame control and use the Frame.Caption property to indicate what should be entered in the text box. Put a Frame control on the form, and put a TextBox control inside the Frame control. Then drag this frame back to the toolbox, and you have a new control that combines these two controls. I really like the ability to add a new page to the toolbox (use the right-mouse button to click on the Toolbox window) and insert my new controls on the new toolbox page. Then I can export this toolbox page and import it into any project (see Figure 1). Take the time to explore the new IDE until you are comfortable with it.

### LEARN TO PROGRAM FOR EXCEL

Now let's get back to writing some code for Excel. The Excel object model is logical and easy to understand. Its objects range from Workbook objects at one end of the spectrum to Range objects at the other. No, there is no Cell object. The lowest-level object is a Range. But because a Range object can contain just one row and one column, you can use the Cells method to create a range of only a single cell. Prove to yourself that this works with this code. In the Code window, type this procedure:

```
Sub TestCell
Dim MyCell as Range
Set MyCell = Selection.Cells(1, 1)
MsgBox MyCell.Rows.Count
```

Then move back to Excel by clicking on the Excel button on the toolbar. Select a few cells with the mouse, then select the Macro menu item from the Tools menu, and run the TestCell macro. You should see the message box that shows the MyCell range has only a single row.

Try another example using a Range object. One of the most common things you will want to do with Excel as you write VBA procedures is add a new row with columns of information to an Excel worksheet. Write a generic AddRow procedure that you can call from all of your routines. To make the procedure generic, you can have it add information to a row beginning with the current active cell. Then, at the end of the procedure, set the active cell to the one directly below this one so you are positioned to call the procedure again with the next row of information.

But how many cells do you want to fill in on this row? In the past, one of the problems with writing such a generic procedure was that you needed to write the procedure to accept a fixed set of arguments, or you needed to concatenate the arguments into a delimited string and parse that string within the procedure. One of the syntax additions to VBA, however, is the ParamArray argument, which allows you to write the procedure to expect an array of parameters rather than a fixed number of parameters. This functionality lets you call the AddRow procedure (see Listing 1) with only the number of parameters that you want to add to the row:

```
AddRow "Date", "Payee", "Amount", "PayDate"
AddRow "01/02/97", "Airo", "34.42", "01/02/97"
AddRow "01/02/97", "Bship", "124.75"
```

Start writing your procedure by defining the array, Args(). Use the keyword, ParamArray, to hold the parameters for the procedure. Then write a For-Next loop that steps through these parameters. You are probably safe to assume that the ParamArray begins with array element zero. In prior versions of VBA, the ParamArray could be zero- or one-based, depending on where it was called from. One of the few changes from Excel 95, however, has been to make all ParamArray calls zero-based. In spite of this, to be completely safe, you should use the LBound

function to get the index of the first parameter. In any case, you need to use the UBound function to get the index of the last parameter:

```
Private Function AddRow(ParamArray Args())
Dim i%
For i = LBound(Args) To UBound(Args)
```

When you call this procedure, the Selection range is placed at the first cell to be filled with the first argument. You place data in the range using its Formula property. Now use the Offset method of the Selection range to move over one column, and use the Select method to select that cell. Continue looping through the array of parameters:

```
Selection.Formula = Args(i)
Selection.Offset(0, 1).Select
'select next col
Next
```

After you exit the loop, the variable "i" contains the number of parameters. This variable also contains the number of columns that you moved from the starting column. Use the Offset method again to move back that number of columns and down one row to prepare for the next call to this procedure:

```
Selection.Offset(1, -i).Select
```

```
Attribute VB_Name = "Module1"
Option Explicit

Public Sub ShowAcctForm()
UserForm1.Show
End Sub

Sub TestCell()
Dim MyCell As Range
Set MyCell = Selection.Cells(1, 1)
MsgBox MyCell.Rows.Count
End Sub

Public Function AddRow(ParamArray Args())
Dim i%
For i = LBound(Args) To UBound(Args)
Selection.Formula = Args(i)
Selection.Offset(0, 1).Select
'select next col
Next
Selection.Offset(1, -i).Select
'select next row
End Function

Public Function GetDTS(H$) As Double
'returns date serial number from hex string
Dim TempH$, pos% 'add pos& check 5/11/94 CRB
On Error Resume Next
TempH$ = Trim(H$) 'CRB 05/05/94
If Len(TempH$) Then
pos& = InStr(TempH$, ".")
If pos& Then TempH$ = Left$(TempH$, pos& - 1)
If Len(TempH$) > 8 Then TempH$ = Left$(TempH$, 8)
GetDTS = (Cdbl("&H" + TempH$)) / 86400 + 25569
If Err = 13 Then GetDTS = 0
End If
On Error GoTo 0 'CRB 1/19/96
End Function
```

**LISTING 1** *Module with Generic Helper Procedures.* Always put your helper procedures in a separate module so you can use them easily in other projects. This source is embedded in the Excel workbook file. Notice the VBA5 Attribute within the module with the procedure description.



## GETTING STARTED WITH VBA

'select next row  
End Function

It's that easy! I won't go into much more detail on Excel's object model in this column. For more details on how to work with the properties and methods exposed by Excel, consult some of my previous columns [September 1995 and earlier] on Excel 5 written before the release of VB4.

### UNLIMITED DATABASE ACCESS

Prior versions of Excel provided limited database access by using ODBC or calls to MSQuery. But now you have the full power of Visual Basic database access at your fingertips. Take a look at a real-life example, while you learn how to use the new forms engine.

SunOpTech's ObjectOrder system has a customization module that interfaces with several different accounting systems to pass invoice and purchase order data. The controller for one of our clients wanted to be able to preview this information in Excel. In the past, we would have written a VB4 program to access the order database, then use OLE Automation to put the information into Excel. With Excel 97, however, you can write the code within an Excel workbook (see Figure 2). With few changes, I took the VB4 procedure and loaded the BAS module into VBA97.

Remember that you need to reference the type library of any ActiveX object that you are going to use in your VBA procedures before you can use its properties and methods in your code, so that VBA can perform early binding.

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 included in this References dialog to validate and compile your code. If you do not include the proper object libraries,

WITH THE RELEASE OF  
OFFICE 97, YOU CAN  
WRITE A ROUTINE IN  
VBA TO CONTROL EXCEL,  
POWERPOINT, WORD,  
PROJECT, OUTLOOK,  
AND ACCESS.

ies, you get syntax errors in your code. For example, if you try to reference the Database or Recordset object without including one of the DAO object libraries, it will seem that VBA does not understand database access. By checking the Microsoft DAO 3.5 Object Library item in the list box, though, you have full access to these objects.

Start with the form. You need to give the user a way to select a range of order

numbers and two buttons—one for Import and one to close the form. Open a new Excel workbook and move to the VBA development environment by pressing the Alt-F11 key combination. Insert a form and draw a Frame control. Then put a TextBox control inside the Frame control. Change the TextBox control's Name property to txtFrom. Copy the Frame and TextBox controls, and name the second text box txtTo. Then add a CommandButton control with the caption Import and another with the caption Cancel. Double-click on the Cancel button to expose the Code window, and enter this code to unload the form:

```
Private Sub butClose_Click()  
Unload Me  
End Sub
```

Then add code to call an Import procedure, and pass the order numbers from the Textbox controls on the form:

```
Private Sub butImport_Click()  
Import txtFrom, TxtTo  
End Sub
```

### BEGIN THE IMPORT PROCEDURE

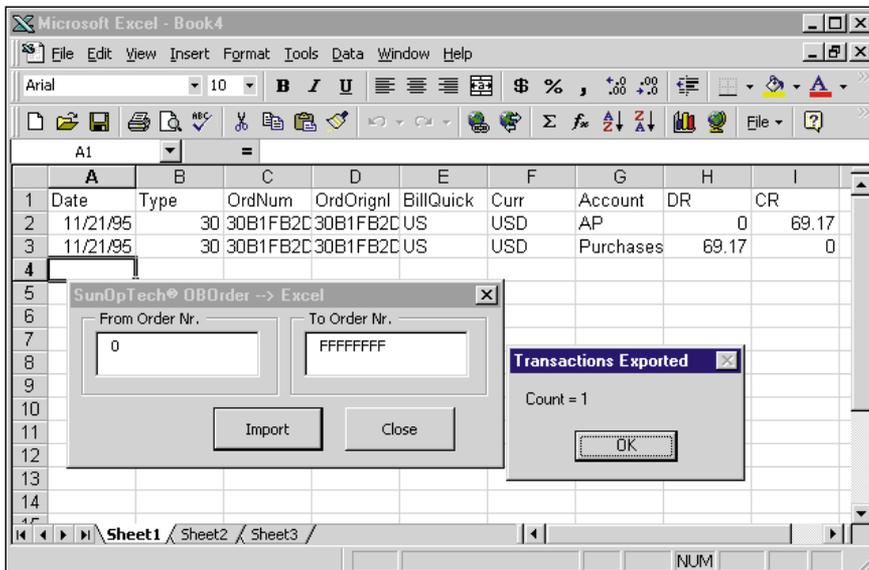
You can download the complete Import procedure from The Development Exchange (for details, see the Code Online box at the end of this article). You can test this code using GS9701.xls, the OBOOrder database application (both files are available in the Registered Level of DevX). Begin by opening the database and the order header table, and setting the index to the order number:

```
Sub Import(ds$, de$)  
Set dbOg = OpenDatabase(DBPath)  
Set rsOHg = _  
    dbOg.OpenRecordset("OrdHeader")  
rsOHg.Index = "OrdNum"
```

Add a new Workbook by calling the Add method of the Workbooks collection. Call the generic AddRow procedure to add a heading line:

```
'now open export file and write  
'header line  
Workbooks.Add  
AddRow "Date", "Type", "OrdNum", _  
    "OrdOrigin1", "BillQuick", _  
    "Curr", "Account", "DR", "CR"
```

Then use the Seek method of the Recordset object. Start at the beginning order number, and begin a Do loop that will terminate when the end of the order header table is reached. Note that the order num-



**FIGURE 2** The Running Application. The new Microsoft Forms engine lets you design forms with any ActiveX control.



## GETTING STARTED WITH VBA

ber index is in descending sequence, so you need to use the BOF property rather than the EOF property. Within the loop, check to see if the last order number has been reached. If so, exit the loop:

```
rsOHg.Seek "<=", ds$
Do While Not rsOHg.BOF
    OType = Trim(rsOHg("Type"))
    ONum = Trim(rsOHg("ORDNUM"))
    If ONum > de$ Then Exit Do
```

Now determine if the order record is a sale or purchase, and call the XLAdd procedure to add the information to the Worksheet. Then move to the previous record:

```
If OType = "20" Then
    'skip type orders
ElseIf LocQuick = Trim(rsOHg_
    ("BFRQuick")) Then    'IsSale
    ALAcct = "AR"
    IEAcct = "Sales"
    TotCnt = TotCnt + XLAdd_
        (ALAcct, IEAcct, OType, ONum, _
        rsOHg("BTOQuick"), False, True)
Else 'IsPurchase
    ALAcct = "AP"
```

```
IEAcct = "Purchases"
TotCnt = TotCnt + XLAdd_
    (ALAcct, IEAcct, OType, _
    ONum, rsOHg("BFRQuick"), _
    False, False)
End If
rsOHg.MovePrevious
Loop
```

The XLAdd procedure returns the number of imported records, so the count can be accumulated and displayed in a message box at the end of the procedure:

```
msg$ = "Count = " & TotCnt
MsgBox msg$, vbOKOnly, _
    "Transactions Exported "
End Sub
```

Review the entire XLAdd procedure on the Registered Level of DevX. It interprets the information in the order record to prepare the appropriate debits and credits, and it summarizes the order detail lines by account with a SQL statement. Once the data is in Excel, you can use VBA to perform any analysis or formatting you can do from the Excel menu. For an expanded version of the code

that prepares an Excel pivot table and account pie chart from the data in the order database, go to the Premier Level of The Development Exchange. ❌

### 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 Excel 97 with VBA Locator+ Codes

*Listings ZIP file and GS0197.zip, which includes the complete Import procedure and the OBOOrder database application (free Registered Level): VBPJ0197*

★ *Listings for this article, the complete Import procedure, the OBOOrder database application, plus an Excel 97 file that helps you prepare an Excel pivot table for the data in the order database (subscriber Premier Level): GS0197P*