Tools Add In Visual Basic for Applications

by Chris Barlow

WRITING USER-DEFINED FUNCTIONS IN VBA

You've just loaded Microsoft's new Excel 5.0 on your computer and you're surfing through the menus to see what's new. You select Module from the Macro submenu of the Insert menu, and your familiar spreadsheet changes to a white screen with a strange new Visual Basic Toolbar! If you're not sure what to do next, you've come to the right place.

It's a little scary when you first see that strange Toolbar. But if you have ever prepared a new spreadsheet in Excel you already know most of what you need to know to get started as a VBA programmer. If you've ever written an Excel macro, you have a real head start. And if you already know Visual Basic, you'll love some of the new features built into VBA.

SOMETIMES THE BEST WAY TO GET SOMETHING DONE IS TO WRITE YOUR OWN PROCEDURE.

Excel 5 is extremely powerful: Microsoft has gone to a lot of trouble to build in the features that most people need. Try the new Pivot Tables to analyze some data, or experiment with Custom Auto-Fill to easily fill a range with your own lists. You can do almost anything you need to do without ever clicking the Module menu item.

But there are times when the best way, and sometimes the only way, to get something done is to write your own *procedure* that does exactly what you want. You don't have to be a programmer to do this. Microsoft has made it easy with the Macro Recorder.

But how do you know if a particular task should be done in VBA? In order to determine this, I usually analyze the task in question in terms of its decision-making characteristics and repetition of functions.

Most complex spreadsheets are full of formulas that make decisions. If you've worked with spreadsheets you've seen those big IF statements that take up the whole width of the screen and require that all the parentheses are in the right spot. Those IF statements are hard to read and hard to debug. I find it much easier to write down what I want to do than to type that huge IF statement.

For example, suppose you have spreadsheet designed to calculate the commission on the sale of an item. Column A contains the item code, Column B contains the units sold, and Column C contains the dollar amount. You need to calculate the commission according to a commission schedule, and put the commission in column D. The commission schedule is:

2% if the amount is less than $100
3% for amounts between $100 and $500
5% for amounts over $500
plus a bonus of $1 for each unit sold
unless the item code begins with a "Y" which gets a $2 bonus.

I'll leave it to you to figure out the complex IF statement you'd need to put into Column D. I prefer VBA because it lets me stay away from those big IF statements. I'll walk you through the process of writing a VBA user-defined function procedure that you could use in Column D as easily as any of Excel's built-in functions. The procedure follows the line-by-line outline of the commission schedule, and uses two different VBA decision-making methods: SELECT CASE, and IF...THEN...ELSE.

First, in the blank Module 1 screen, define the function, including parameters that will be passed. Add a comment line beginning with an apostrophe so you'll know later what this function does:

```
Function CalcCommission(Item, Units, Amount)
'This function calculates commissions
```

Next, add the routines to calculate the portion of the commission based on amount, and store the calculation in a variable called AmountComm. A SELECT CASE statement executes one of several groups of statements, depending on the value of an expression. The statement determines the amount in which CASE fits, and then performs the calculation only for that CASE:

```
'first calculate the Amount Commission
Select Case Amount
Case Is < 100
   AmountComm = Amount * 0.02
Case 100 To 500
   AmountComm = Amount * 0.03
Case Else
   AmountComm = Amount * 0.05
End Select
```

Now you can calculate the portion of the commission based on units, and store the calculation in a variable called UnitComm. The IF...THEN...ELSE statement conditionally executes a group of statements, depending on the value of an expression. It evaluates the IF expression to see if it is true. If the expression is true, it performs the next statement. Otherwise it skips to the statement after the ELSE statement:

```
'now calculate the Unit Commission
If Left$(Item, 1) = "Y" Then
   UnitComm = Units * 2
```

Chris Barlow learned Basic with Professors Kemeny and Kurtz at Dartmouth College. He develops Visual Basic applications for Sun Hydraulics' SunOpTech Group, and with co-author Ken Henderson he wrote an application that was a finalist in the first Windows Open Contest. Contact Chris at Sun Hydraulics—SunOpTech Group, 1500 West University Parkway, Sarasota, Florida, 34243; or by phone, 813-351-9183; fax, 813-355-4497; or CompuServe, 76446,1370.

```
Else
  UnitComm = Units * 1
End If
```

The final step is to set the name of the Function equal to the total of AmountComm and UnitComm. This returns the calculated commission into the proper cell:

```
'now return value in Function name
CalcCommission = AmountComm + UnitComm
End Function
```

Your first user-defined function is complete and ready to insert into your worksheet. Move to Column D, click on Insert and select Function from the menu to pop up the Function Wizard (see Figure 1). The new function is listed along with Excel's built-in functions as a user-defined function. When you select CalcCommission, Function Wizard will help you fill in the parameters one at a time—just click on the appropriate cell on the worksheet.

The easiest way to understand what these statements do is to watch them execute in the code window. Move to Module 1 and put the cursor on the first line of the CalcCommission function. Press F9 to set a breakpoint, which will stop the execution at that point. Return to the worksheet and press F9 again to recalculate the worksheet. The code window will reappear with the cursor stopped at the breakpoint.

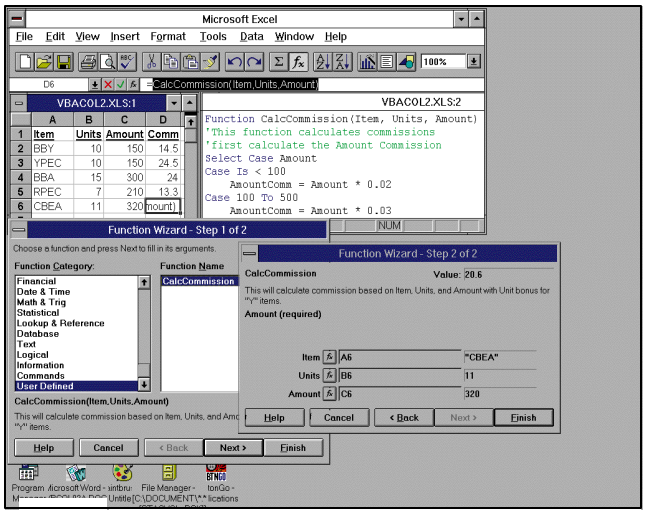
Now Press F8 to single-step through the procedure. You can see the value of any variable by placing the cursor on that variable and pressing Shift-F9. You'll be able to see how different groups of statements are executed based on the value of the conditions.

Whenever you need to change a result based on different conditions, user-defined functions are much easier to work with than a long IF statement. Take a look at the code for the CalcCommission function and compare it to an IF statement that does the same thing. Which is clearer? Which would you rather pick up six months from now to make a correction?

PLAY IT AGAIN

The second thing I look for when deciding to write a VBA procedure is repetition. Do I need to perform the same steps over and over? Programmers call this looping, and there are three different sets of VBA statements you can use to stop the loop, depending on how you want it stopped.

The FOR...NEXT statement is the simplest loop. It will repeat a group of statements a certain number of times. Try typing this in Module 1:



Rely on Wizards. Notice how the Function Wizard shows the return value after you fill in the arguments.

```
Sub TestFORLoop()
For i = 1 To 10
  MsgBox CStr(i * i)
Next
End Sub
```

Now place the cursor on the first line of this procedure and press F5 to run the procedure. You'll see ten message boxes showing the squares of the numbers from 1 to 10.

The DO LOOP and WHILE...WEND statements are usually used to repeat a group of statements while checking for a certain condition. The DO LOOP repeats a block of statements while a condition is true or until a condition becomes True. The WHILE...WEND statement executes a series of statements as long as a given condition is true.

Type the following in Module 1:

```
Sub TestLoops()
Do
  i = i + 1
  MsgBox CStr(i * i)
Loop Until i = 5

While i < 10
  i = i + 1
  MsgBox CStr(i * i)
Wend
End Sub
```

Now place the cursor on the first line of this procedure and press F5 to run the procedure. You'll see another ten message boxes—the first five from the DO loop and the second five from the WHILE loop.

Often these loops are combined with the SELECT CASE or IF...THEN...ELSE conditional statements to check each cell in a range for a certain value, and take action based on that value.

For example, Excel 5 now offers rich text formatting of cells: you can combine different fonts within a single cell. Excel 5 also lets you set up custom number formats for cells. But these custom formats can't make use of this rich text formatting. You can write a procedure that will provide a custom format for special circumstances so that you can learn more about conditions and loops.

Suppose you want to check a selected range of cells, boldface any Commissions that are greater than 20, and add the word "Note" in red underlined italics to the first column as an indication that these items should be reviewed. Start by listing what you need to do:

```
check the Commission column in each row in a selected range
if the Commission > 20 then
    make the commission Bold and add Note to the Item
then loop back and do the next row
```

The first step is to determine how Excel makes the Commission font bold and adds the red italicized and underlined word to the first column. You could search through the Excel manuals and help file to figure out the proper code, but I always let the Macro Recorder do the translating for me.

Select New Macro item from the Macro Record submenu of the Tools Record menu. Name the new macro "FigureOutCode" and click on "OK".

Move to the Comm column and select Font=Bold from the Format Cells submenu. Then move to the Item column and type "Note," highlight it with the mouse, and select Font=Italic, Underline=Single, and Color=Red from the Format Cells submenu. Click on the Stop Recording button and look at the generated code, which is excerpted below:

```
Sub FigureOutCode()
  Range("D6").Select
  With Selection.Font
```

```
    .FontStyle = "Bold"
  End With
  Range("A6").Select
  ActiveCell.FormulaR1C1 = "CBEA Note"
  With ActiveCell.Characters_
                        (Start:=5, Length:=5).Font
    .FontStyle = "Italic"
    .Underline = xlSingle
    .ColorIndex = 3
  End With
End Sub
```

Doesn't the Macro Recorder make it easy? The Macro Recorder even uses the WITH-END WITH statements to make the code easier to read and faster to execute.

You can see how to apply rich text formats to a cell, and you have some code that you'll be able to paste into your new procedure.

Now you can start writing the procedure to perform special formatting. In Module 1, move the cursor below the CalcCommission function and type:

```
Sub CheckCommInRange()
'this marks a selected range of Item,Units,
'Amount,Comm to highlight

'  items with high commission

'first put the selected range in an object
Set CheckRange = Selection
```

Because you want to perform an action but you don't need to return a value, you will use a Sub procedure instead of a Function procedure. Also, you must assign the current selection to an object to use in the loop.

The repetition in this procedure comes from a FOR...NEXT loop that steps through each row in the selected range, and uses the user-defined function CheckComm to make a decision.

If the Commission is greater than 20, the loop will call another user-defined procedure, HighlightRow, to do the formatting:

```
'then begin our loop to go through each row

For Each CheckRow In CheckRange.Rows
  'decide whether this row needs highlighting

  If CheckComm(CheckRow.Cells(1, 4), 20) Then
    'perform special highlighting for this row
    HighlightRow CheckRow
  End If
Next
'now go back and repeat for each row
End Sub
```

Although this could be done in a single procedure, it is common to break such actions into different procedures to make them easier to understand.

The next step is to add the two new user-defined procedures. You'll use a function procedure for the first procedure because a value must be returned. The CheckComm function returns true if the commission is over a given value.

The function receives two parameters: a cell object so it can check its Value property, and the upper limit for commissions. If the commission is greater than the parameter Over, then the value of this function will be true and this row should be highlighted:

```
Function CheckComm(CheckCell, Over)
'this returns True if this cell is
'over the given value
If CheckCell.Value > Over Then
  CheckComm = True
Else
  CheckComm = False
End If
End Function
```

The final step is to write the procedure to highlight a row. The Macro Recorder has given you most of the code for this procedure—simply change the code to use your variables.

In this case, a Sub procedure will perform an action and pass the procedure to a Row object called HRow as its only parameter:

```
Sub HighlightRow(HRow)
'this bolds the Commission and adds Note to Item
'first set Comm column to Bold
HRow.Cells(1, 4).Font.FontStyle = "Bold"
'then get length of Item
StartPos = Len(HRow.Cells(1, 1).Value) + 1
'add word "Note" to Item
HRow.Cells(1, 1).Value = HRow.Cells(1, 1).Value _
                    & " Note"
'now do rich text format of just the added characters
With HRow.Cells(1, 1).Characters(Start:=StartPos, Length:=5).Font
  .FontStyle = "Italic"
  .Underline = xlSingle
  .Color = RGB(255, 0, 0)
End With
End Sub
```

Now that your procedures are complete, you can test them. Move to the worksheet and use the mouse to select several rows. Then select Macro from the Tools menu and click on CheckCommInRange. Click on Run to run the procedure, or click on Step to watch the execution line by line.

I've used both Sub and Function procedures in VBA to do some things that would be difficult or confusing to do in Excel. Each of these procedures makes decisions based on the value of certain expressions.

Now you know how to repeat certain groups of statements in procedure loops. This is a good start on the road to becoming a VBA programmer. I've uploaded the WorkBook with these procedures to the Excel forum on CompuServe. Download it, give it a try, and send me your questions and suggestions. n