Rohit Garg

# Visual Basic for Applications

28-Oct-10

The report has been designed to give a brief overview of the various features of VBA.

1. **Macro recorder:** VBA has a built in recorder that has the ability to record your actions in your spreadsheet and translate them into VBA. However, you cannot do everything with the macro recorder. Your actions are limited to the things you are not able to do in excel.

2. **Recording macros:** Open excel, and activate the macro recorder by choosing Tools →Macro →Record →New Macro. Choose a name for the new macro in the dialog and press OK. Type something into a few cells and stop the recorder. You have recorded a macro. To test if it works, run the macro. This is done by choosing Tools →Macro →Macros. You will notice that the macro enters the text or numbers in the exact same cells as you did.

3. **Recording with relative and absolute references:** In relative referencing the data is placed with respect to the position of the cursor. Start the macro recorder and press the relative references button. Now fill in some information in the cells and turn the macro recorder off. If you try to run the macro now, you will notice that it inserts whatever information you have told it to, but it does it where the cursor is pointing. In absolute referencing the data is placed in the pre-defined cells irrespective of the position of the cursor.

4. **What can be recorded:** The macro recorder can record basically any task you can do yourself in Excel. However VBA has lots of other abilities that you cannot do yourself in Excel, some of them being automating tasks, looping, creating dialog boxes and custom menus.

5. **Assign a macro to a shortcut key:** Open the macro dialog box and mark the macro you are interested in and click Options. A dialog box opens and you can enter a shortcut. Ctrl+ is already filled in and you can enter letter in the little box.

6. **Assigning macros to menu buttons:** Right-click the menu bar, choose customize. Click the Commands tab in the dialog box. Find Macros in the Categories section. Drag the custom button to the menu bar. Click modify selection in the dialog box. Choose assign macro, and assign the macro. Click Modify selection again and choose Edit button to edit the image in the button.

7. **The Visual Basic Editor:** MS Office Suite comes with a built in programming environment, The Visual Basic Editor. The Project browser gives you a brief overview of the workbooks that are open at the moment and what code they contain. The modules are the default place to insert code for the macro recorder, and the macros you have recorded are placed in the modules in this folder.

8. **The object model:** All macro programming in VBA is based on an object model, and is thus called object orientated programming. Usually objects are considered to be things that you can touch and look at. Here we will define an object as a part of a virtual environment with certain properties and functions. VBA is actually just a tool to access and manipulate the different objects of Excel.

9. **Excel objects:** The mother object in Excel is called Application. This is a direct reference to the Excel program itself. An Excel program can contain several open Workbooks. Each of these Workbooks contains a number of sheets, and these contain cells. Excel can also deal with implicit references. If you do not state the fully qualified reference, VBA fills in the missing part of the reference based on the context. For instance:
   *Range ("A1").Value = 68*
   *Means the range A1 in the active sheet of the active workbook is set to 68.*

10. **The object browser:** VBE has a built-in tool for locating objects and properties. It is activated by pressing F2 in the VBE. The object browser basically allows you to find information in two ways. By searching or by stepping through a hierarchy.

11. **Variables and constants:** A variable is an identifier for a storage location which holds a value of some sort. Constants are just variables that cannot be changed.

12. **Data types, declarations and scope:** A variable is a name that contains a reference to a value. Variables can have different data types. If you introduce a variable name in your code without any prior declaration of the variable, VBA will automatically assume that this variable is of type Variant. This means that it can contain any type of data and VBA will automatically change the type of the variable when needed. The declaration of variables is done with the Dim statement. The declarations must be made before the code where the variable is used.

```
Dim MyNumber As Integer
Dim MyName As String
```

13. **Scopes in VBA:** There are in principle 3 different scopes in VBA.
    - *Procedure:* Variables with procedure level scope have their declaration within a procedure. They disappear when the macro ends.
    - *Module:* Module level scope variables are declared at the beginning of the module before any macro code. They are visible to all macros in that module, and they will keep their values even after a certain macro has ended.
    - *Global:* Global or public variables are visible to all modules in the workbook, and should be declared with a Public statement at the module level.

14. **Object variables:** Object variables are variables containing a reference to a certain object. This could be a sheet, a chart, a range etc. The object variable would have the same properties and methods as the object itself. To assign an object to an object variable you need to use the Set statement.

```
Sub ObjTest()
Dim theRange As Range
Set theRange = ActiveSheet.Range("A1:C25")
End Sub
```

15. **Functions and subroutines:** A function is a structure that requires some sort of data and based on this data, it performs a series of operations and finally returns a value. For instance “Average” function takes a series of numbers as its arguments and returns the average of these values. Some functions need several arguments, others can work without arguments. Subroutines, procedures and macros are the same thing. A subroutine is a procedure, and the subroutine can be executed as a macro in Excel.

16. **Declaration of a macro:** The Sub keyword declares the macro and it is followed by the name of the macro. The statements that should be executed by the macro follow the sub declaration and the macro ends with an End Sub statement. The following shows the declaration of a macro:

```
Sub MacroName()
[Statements]
End Sub
```

17. **GoTo statement:** The GoTo allows you to jump to labels you insert in the code. You should avoid using the statement because your code becomes very messy and difficult to read, and you tend to get unexplainable behavior in you macros, because you lose track of the program flow.

```
GoTo MyLabel
[Statements]
MyLabel:
[Statements]
```

18. **If-Then-Else statements:** If statements are the most widely used control statements in VBA. In the simplest form it is very easily understood, but nested if statements can get complicated. The basic form of the if statements is:

```
If [Condition] Then
[Statements]
Else
[Statements]
End if
```

19. **Select Case structure:** When you need VBA to choose between lots of different options, the Select Case structure is a better solution. The general form is:

```
Select Case [Expression]
Case [Expression]
[Statements]
Case [Expression]
[Statements]
Case Else
[Statements]
End Select
```

20. **For-Next loop:** The loop structure allows you to ask VBA to repeat the same task for a certain number of times. This cannot be done with the recorder. The most widely used loop structure is the For-Next loop. This structure uses a counter to determine the number of times a certain set of statements have been executed.

```
For counter = startValue To endValue [Step stepValue]
[Statements]
Next counter
```

The step value is optional. If no step value is provided, the counter value is incremented by one on each run through the loop. If for some reason you want to exit the loop based on a decision include "Exit For" statement.

21. **Do-While:** Another example of a loop structure is the Do-While.

```
Do While [Condition]
[Statements]
Loop
```

22. **Error handling:** GoTo statement can be used to handle errors. The errors in question are the so-called intentional error. You ask VBA to perform a loop, knowing that sooner or later it will fail for some reason. When that happens you use a GoTo statement to exit the loop without the whole macro failing.

```
On Error GoTo NoMoreFiles
[Statements]
NoMoreFiles:
MsgBox "There are no more files"
```

23. **Referencing ranges:** Ranges can be referenced in a number of different ways. The simplest way is to use the address of the range.

```
Range("A1:B6")
```

24. **Value property of Range objects:** The value of a cell can be read with the Value property. You can set the value of several cells at the same time with the value property of a multi-cell range.

```
Range("C3").Value = 10
Range("C3:K100").Value = 1000
```

25. **Count property of Range objects:** The number of cells in a range can be counted. Alternatively the rows or columns can be counted.

```
Range("C3:K100").Count
Range("C3:K100").Rows.Count
```

26. **Font and Interior property of Range objects:** Font and Interior properties return Font and Interior objects. These allow you to do formatting on cells.

```
Range("A1").Font.Bold = True
Range("A1").Interior.Color = RGB(255,0,0)
```

27. **Formula property of Range objects:** Ranges have a Formula property. This property can be manipulated like most other properties.
    *Range("A1").Formula = "=AVERAGE(A1:C17)"*

28. **Creating a chart object:** A chart can exist in two different ways in excel. The chart can either be a standalone sheet with nothing but a chart on it or it can be a ChartObject embedded in a sheet. To create a standalone start you declare a Chart variable and then create the instance of the chart. To create the embedded chart you need to create a new ChartObject in the sheet, and access the chart through this ChartObject. Elements in the chart of the chart object can now be accessed through the ChartObject.

29. **Creating UserForm:** A UserForm is a custom-built dialog box that you build using the Visual Basic Editor. With a UserForm you can create a user-friendly interface for your document, making data entry more controllable for you and easier for the user.

30. **Insert a new UserForm:** Make sure that the current workbook is selected then open the Insert Menu and choose UserForm. When you do this a new, blank UserForm appears in the code window of the Visual Basic Editor and a corresponding entry appears in the Project Explorer.

31. **Rename the UserForm:** With the UserForm selected find the Name property in the Properties Window and change it. The Project Explorer now displays the UserForm's new name. When naming forms and their controls remember that you must not include spaces in the name, or use any of the VBA "reserved words”.

32. **Add a TextBox:** Add a TextBox control to the form by clicking on the TextBox button in the Toolbox then clicking somewhere near the centre of the form. You can change the size and shape of a control either by dragging the resizing or by changing the values of its Height and Width properties in the Properties Window. Each control should have a meaningful name so that when you write the code you can easily identify it.

33. **Add a Label:** Use the toolbox to place a Label control on the form. To change the caption of the label you can either type directly on to the label or you can change its Caption property in the Properties Window.

34. **Create the ComboBox:** The ComboBox needs to be told where to get the information to build its list. There are two ways to do this. It can be done with code or it can refer to a named range of cells in the workbook. Switch to Excel and open a worksheet in the same workbook. Type a column of items representing the entries you want to appear in the ComboBox list. Now select the cells containing the list items and name the range of cells. Return to the Visual Basic Editor and click on the ComboBox to select it then go to the Properties Window and find the RowSource property. Enter the same name as you used for the range containing your list.

35. **Check the Tab Order:** Many people like to move around from control to control by clicking the Tab key on their keyboard. To change the order, close the form and in the Visual Basic Editor open the View menu and choose Tab Order. Here you can move items up and down the list to control the behavior of the key in the form.

36. **Write the VBA Code:** After finishing the design of the form, the next job is to write the VBA code to power it. The code is needed to power the command buttons. Double-click the command button to open the UserForm's code module. It is easy to open the UserForm from the Visual Basic Editor, but the person who is going to use this tool needs an easy way to open the form from Excel. The user can run this macro from the menu in the usual way Tools → Macro → Macros or you could assign it to a custom menu item or a button on the worksheet.
    *Unload Me: closes the UserForm*
    *UserForm_Name.Show: opens the UserForm*