“The VBA Crash Course”

By Erik Loebl

http://yourexcelvbatraining.com
# The VBA Crash Course

## Table of Contents

1. VBA Introduction  
2. VBA Message Box  
3. VBA Input Box  
4. VBA Cells  
5. VBA Procedures and Scope  
6. VBA Left Function  
7. VBA Right Function  
8. VBA Mid Function  
9. VBA Trim Function  
10. VBA Concatenate  
11. VBA Len Function  
12. VBA Instr Function  
13. VBA CINT Function  
14. VBA If Then Statement  
15. VBA Select Case Statement  
16. VBA For Next Statement  
17. VBA Do Loop Statement  
18. VBA Read Text Files  
19. VBA Read Excel Files  
20. VBA Error Handling  
21. Putting it all together
VBA Introduction

Hi, I’m Erik, I have been working with the various forms of VBA for over 10 years. I can recommend the information you are going to be learning in this short manual. In this report you are going to find the items that I think comprise the basics of what you need to know to start with VBA, and to do your job better. It is like a **VBA crash course.** I use this information nearly every day, and you are going to find out the fundamentals that you need to know.

The goal of this report is to show you the aspects of VBA that you will most likely need for you job. It is not my intent to go into the details of every function in this report. There are many resources that do this, and you can glean this extra information from them. Please take a look at my website, [http://yourexcelvbatraining.com](http://yourexcelvbatraining.com) for some of these recommended resources.

VBA (Visual Basic for Applications) is a rapid application development (RAD) platform. It allows you to build intelligent apps fast. You are going to find different forms of VBA for all the Microsoft applications. They are all basically the same, except for the object libraries they expose (for example Excel VBA exposes Excel specific objects and Access exposes Access objects, etc.). The form of the VBA language pertains to the application it is in. You are going to find that the basics covered in this manual can be used in whatever Microsoft application you are using. Lastly, VBA (or any programming) is based on **Algebra, assigning a value to a variable.**
Now onto the learning (You can copy and paste any of these examples in your VBA Editor and run them to see them work.)....

First Open Excel and access your Visual Basic Editor (VBA editor) by pressing the ALT + F11 keys on you keyboard. Or you can find it near the “Macro” button.
VBA Message Box

The VBA Message Box is used to alert the user of both the good and bad. They may have gotten their password correct, may have complete a long waiting process, or the may have entered the wrong value, or they may have encountered a critical error that you need to know about. These are great situations to use the message box in.

If you have never created a message box before, do this…

Basic

Public Sub FirstMsgBox()
    MsgBox "Hi user!"
End Sub

More Advanced

In the following example, the user gets a critical error message when the strFileName variable is empty…

Public Sub ReadTextFile()
    If strFileName = "" Then
        MsgBox "No File was selected.", vbCritical, "No File"
        Exit Sub
    End If
End Sub

Note: Press the F5 key on your keyboard when your cursor is between the “Sub” and “End Sub” parts to run your code. Press the F8 key on your keyboard to advance the code line by line.
**VBA Input Box**

The input box accepts data like some text or a value to be entered. The entry is then evaluated and responded to based on the result.

*(Please note that when I put the apostrophe in front of a line of text, the text or the line gets “rem’d” out, it doesn’t get run with the rest of the code.)*

**Basic**

```vba
Public Sub InputBox()
    Dim strNumber As String
    'Show an inputbox to accept a numeric value
    strNumber = Application.InputBox("Enter A Number")
    'Show the user a message based on their entry…
    MsgBox strNumber
End Sub
```

**More Advanced**

The following code evaluates the entry of the user and sends back a result.

```vba
Public Sub InputBox2()
    Dim strNumber As String
    Dim dblNumber As Double
    Dim strMessage As String
```

Note: Press the F5 key on your keyboard when your cursor is between the “Sub” and “End Sub” parts to run your code. Press the F8 key on your keyboard to advance the code line by line.
Show an inputbox to accept a numeric value, and put the entered value into a string datatype…

strNumber = Application.InputBox("Enter A Number", "Number", 0)

‘Use the IsNumeric function to validate that a numeric value was entered…

If IsNumeric(strNumber) Then
‘Convert the string value to a double datatype…

dblNumber = CDbl(strNumber)

Else
‘If a numeric value was not entered, give it the value of 9999…

dblNumber = 9999

End If

‘Now evaluate the number…

Select Case dblNumber

Case 0 To 1
   strMessage = "Between 0 and 1"

Case 2 To 3
   strMessage = "Between 2 and 3"

Case 9999
   strMessage = "Error"

Case Else
   strMessage = "Greater than 3"

End Select

‘Show the user a message based on their entry…

MsgBox strMessage

End Sub
**VBA Cells**

The cells in VBA can be accessed in 2 ways, with a **range** reference, or a **cell** reference. Since the **range** reference is easier to remember, I would suggest this method for most situations.

The **Dim** statement is used to allocate a certain portion of the computer’s memory to a variable.

We access the objects on the worksheet using the object reference model:

*Worksheets(“WorksheetName”).CellReference=value*

---

**Basic**

```vba
Public Sub WriteToCells()
    Dim strName as String
    strName = “Name”
    ‘Both of these lines of code do the same thing. In my opinion using the “Range” version is easier to remember...
    Worksheets(“Sheet1”).Range(“A1”) = strName
    Worksheets(“Sheet1”).Cells(1, 1) = strName
End Sub
```

Note: Press the F5 key on your keyboard when your cursor is between the “Sub” and “End Sub” parts to run your code. Press the F8 key on your keyboard to advance the code line by line.
VBA Procedures and Scope

Sub Procedures can Public or Private. By default, all subs not specified as Public or Private are considered Public. Sub procedures make up the procedure contained in a module.

In the following example, I added a command button from my forms toolbar and the “Assign Macro” box appears. I can only choose sub procedures which are either labeled as “Public” or not specified (regular subs).

Here is the code (After assigning the Macro, click “Button5” to run code):

**Basic**

```vba
Public Sub NewMsg_public()

'Allocate a portion of memory for a string (text - up to 255 characters) holding variable
```
Dim strMsg As String
strMsg = "Press enter to continue"
MsgBox strMsg
End Sub

Sub NewMsg_sub()
Dim strMsg As String
strMsg = "Press enter to continue"
MsgBox strMsg
End Sub

Private Sub NewMsg_private()
Dim strMsg As String
strMsg = "Press enter to continue"
MsgBox strMsg
End Sub
VBA Left Function

The VBA Left Function allows you to extract the left “X” number of characters from a string.

I normally use this in combination with the INSTR function to extract the first name from an entire name. Use the INSTR function to find the 1st space and then use the VBA Left function to extract the data up to that first space.

Basic

Public Sub GetFirstName()
    Dim strFirstName As String
    'Get the name from a cell in the activated worksheet…
    strFirstName = Range("A2")
    'Extract the 1st 3 characters from the text…
    strFirstName = Left(strFirstName, 3)
    'Write the result to cell B2…
    Range("B2") = strFirstName
End Sub

More Advanced

'This is the example with the INSTR function (this code is more complex, but all of the functions are covered in this manual)

Public Sub GetFirstName2()
    Dim strFirstName As String
    Dim intPos As Integer
    strFirstName = "John Everett Maxwell" 'The value of the name
    intPos = InStr(1, strFirstName, " ") 'Find the position of the first space
strFirstName = Left(strFirstName, intPos) 'Extract the left portion of the text up to the first space

MsgBox strFirstName 'Show the new value

End Sub
VBA Right Function

The VBA Right Function allows you to extract the right “X” number of characters from a string.

I normally use this in combination with the INSTR function to extract the last name from an entire name.

Basic

Public Sub GetLastName()
    Dim strLastName As String
    strLastName = Range("A2")
    strLastName = Right(strLastName, 8)
    strLastName = Trim(strLastName)
    Range("C2") = strLastName
End Sub

More Advanced

' This is the example with the INSTR function (this code is more complex, but all of the functions are covered in this manual)

Public Sub GetLastName2()
    Dim strLastName As String
    Dim intPos As Integer
    Dim intPos2 As Integer
    Dim intLastName As Integer
    Dim intStringLength As Integer
    strLastName = "John Everett Maxwell"
    ' Find the total length of the string...
    intStringLength = Len(strLastName)
End Sub
'Find the position of the 1st space...
intPos = InStr(1, strLastName, " ")

'Increment the position by 1...
intPos = intPos + 1

'Now find the position of the 2nd space using
'the position of the first space...
intPos2 = InStr(intPos, strLastName, " ")

'Find the amount to extract by subtracting
'the position of the 2nd space from the total string length...
intLastName = intStringLength - intPos2

'Now find the last name...
strLastName = Right(strLastName, intLastName)

MsgBox strLastName
End Sub
**VBA Mid Function**

The VBA Mid function allows you to extract the middle “X” number of characters from a string, or all the characters from a certain point.

I normally use this in combination with the INSTR function to extract the last name from an entire name.

**Basic**

```
Public Sub GetMiddleName()
    Dim strMiddleName As String
    strMiddleName = Range("A2")
    strMiddleName = Mid(strMiddleName, 9, 3)
    Range("C2") = strMiddleName
End Sub
```

**More Advanced**

‘This is the example with the INSTR function (this code is more complex, but all of the functions are ‘covered in this manual)

```
Public Sub GetMiddleName2()
    Dim strMiddleName As String
    Dim intPos As Integer
    Dim intPos2 As Integer
    Dim intCharacters As Integer
    Dim intStringLength As Integer
    strMiddleName = "John Everett Maxwell"
    'Find the position of the 1st space...
    intPos = InStr(1, strMiddleName, " ")
    'Increment the position by 1...
```
intPos = intPos + 1

'Now find the position of the 2nd space using the position of the first space...
intPos2 = InStr(intPos, strMiddleName, " ")

'Subtract the 2 values...
intCharacters = intPos2 - intPos

'Use the result to find the middle name...
strMiddleName = Mid(strMiddleName, intPos, intCharacters)

MsgBox strMiddleName

End Sub
**VBA Trim Function**

The VBA Trim function allows you to “trim” the unwanted characters from the beginning and ending of a text string. This is great to remove blank spaces from text values. For example, “Betty” (with space) will become “Betty” (no spaces).

**Basic**

```vba
Public Sub GetLastName()
    Dim strLastName As String
    strLastName = Range("A2")
    strLastName = Right(strLastName, 8)
    strLastName = Trim(strLastName)
    Range("C2") = strLastName
End Sub
```

In this example, any spaces in the front and back of the strLastName string are removed.
**VBA Concatenate**

Concatenating is the act of joining 2 or more strings together. Like you can glue a piece of static (unchanging) text before a variable, like

“The total is “ ___________

You can also use concatenation when you do a mail merge. You can “glue” the first name, middle, and last name together. Or join the city, state, and zip code together. The choice is up to you.

**Basic**

‘In this example, I extract the 1st 3 numbers from a telephone number and append the words “ area code” to the new value…

Public Sub WriteAreaCode()
    Dim strText As String
    strText = Range("A2")
    strText = Left(strText, 3)
    Range("B2") = strText & " area code"
End Sub

‘In this example I join the first and last names together…

Public Sub WholeName()
    Dim strFirstName As String
    Dim strLastName As String
    Dim strName As String
    ‘GET THE FIRST AND LAST NAMES...
    strFirstName = Range("A3")
    strLastName = Range("B3")
    ‘CLEAN THE VARIABLES UP....
strFirstName = Trim(strFirstName)
strLastName = Trim(strLastName)

COMBINE THE 2 VARIABLES...
strName = strFirstName & " " & strLastName

WRITE THE END RESULT...
Range("C3") = strName

End Sub
VBA Len Function

The Len function returns a numeric value which is the length of the string.

You can see that I have used this function already in my previous examples.

I use it to determine the “odd ball” value in a group. If 99 of them are 7 characters in length, and one of them is 8, I can pick it out.

I also use it as a part of other procedures like the left, mid, or right functions. (See the examples)

Basic

Here is the syntax:

Public Sub LenString()
    Dim intLen As Integer
    Dim strName As String
    strName = Range("A1")
    intLen = Len(strName)
    MsgBox "the length of the name is " & intLen
End Sub
VBA Instr Function

This function allows me to see if a certain value exists within a string. Many times I have used this to find certain values that need changing. I can loop through a number of cells and mark the cells that contain the certain value. Then I can just focus on finding and replacing in those cells.

The pseudo code logic is:

1. Loop through a range (see the For Next example)
2. If cell contains a certain value write an “X” to the cell next to it.
3. Sort by the “X”’s
4. Focus on those cells.

Basic

‘Here is some sample syntax of another sample:

Public Sub FindValue()
    Dim intPos As Integer
    Dim strName As String
    strName = Range("A1")
    intPos = InStr(1, strName, "ca")
    MsgBox "The h is in position " & intPos
End Sub

Also take a look at my left, mid, and right function examples.
**VBA CInt Function**

This function a convert type of function. If you would like to convert your function from one type format to another, you'll use this.

Integer values cannot hold decimals. If you have the number 36.32, you may want to store the 36 in an integer value, and the .32 in a decimal so you can evaluate it.

I have done the prior example when I wrote a program to evaluate the decimal and round the result to a certain point.

**Basic**

```vba
Public Sub ParseNumber()
    Dim dblNumber As Double
    Dim intNumber As Integer
    dblNumber = Range("A1") 'Take a decimal number
    intNumber = CInt(dblNumber) 'Put the decimal number into an integer value
    MsgBox intNumber
End Sub
```
**VBA If Then Statement**

The If Then statement evaluates 2 or more conditions and based on the condition a conclusion is made.

In the following code, a name is checked for the presence of a specific name and a message is returned based on this.

**Basic**

```vba
Public Sub IfThen()
    Dim strText As String
    strText = "Sue James"
    If InStr(1, strText, "James") Then
        strMessage = "Found"
    Else
        strMessage = "Not Found"
    End If
    MsgBox strMessage
End Sub
```
**VBA Select Case Statement**

The Select Case statement is easier to read than the If Then syntax. It essentially controls program flow and evaluates different situations.

In the following code the variable `dblNumber` is evaluated and a message is formulated. This static value of 3.25 could be substituted for a cell value in a range. (See the For Next section).

**Basic**

```vba
Public Sub SelectCase()
    Dim dblNumber As Double
    Dim strMessage As String

    dblNumber = 3.25

    Select Case dblNumber
        Case 0 To 1
            strMessage = "Between 0 and 1"
        Case 2 To 3
            strMessage = "Between 2 and 3"
        Case Else
            strMessage = "Greater than 3"
    End Select

    MsgBox strMessage
End Sub
```
VBA For Next Statement

This is used to do something for a number of times. Like to move over a range of cells, and evaluate each cell value. If you need to exit a For Next statement, use Exit For.

Basic

The following code loops through every cell from row 1 to 65,536 in column “A” and enters a value…

Public Sub ForNext()
    Dim lngRow As Long
    For lngRow = 1 To 65536
        Worksheets("Sheet1").Range("A" & lngRow) = lngRow
    Next
End Sub

Sometimes you may need to do the loop in reverse. This code does the same thing in reverse…

Public Sub ForNext_Backward()
    Dim lngRow As Long
    For lngRow = 65536 To 1 Step -1 'The Step -1 syntax is the key to moving backward.
        Worksheets("Sheet1").Range("A" & lngRow) = lngRow
    Next
End Sub
**VBA Do Loop Statement**

You can use the Do Loop statement to loop continuously through a certain situation until the condition is true. If you need to exit a Do Loop statement, use Exit Do.

**Basic**

In the following code the intCounter variable is incremented until it becomes 25.

```vba
Public Sub DoLoop()
    Dim intCounter As Integer
    Do Until intCounter > 25
        intCounter = intCounter + 1
        Loop
End Sub
```

You may need to implement the do loop while the password is wrong. If the password is incorrect keep on entering it until it is correct. You could also limit their number of entries to 3 and then lock the user out.

I have used do loops to increment a certain value while another condition was not yet met.
VBA Read Text Files

As a part of the design of a program, you may need to read in the values from a text file not using the Excel import feature. I am going to show you how to implement this. The following code reads in "C:\FileName.txt" that contains names and address information and is in **tab delimited format** and places the results in a Worksheet called “Sheet1”.

**More Advanced**

Here also is an example of error handling.

```vba
Public Sub ReadTextFile()
    Dim strFileName As String
    Dim intRow As Integer
    Dim strLineItem As String

    On Error GoTo errHandler 'If there is an error go to the errHandler label

    strFileName = "C:\FileName.txt" "This is a variable to hold the path name

    If strFileName = "" Then 'Check and see that the file name is not blank. Give a message if it is…
        MsgBox "No File was selected.", vbCritical, "No File"
        Exit Sub
    End If

    'Clear the worksheet and get it ready for new data…
    Worksheets("Sheet1").Select
    Cells.Select
    Selection.Clear
    Range("A1").Select

    'Prepare the destination worksheet by writing column names on the first row of “Sheet1”
```
```
intRow = 1 'print col headings on first row
Worksheets("Sheet1").Cells(intRow, 1) = "Name"
Worksheets("Sheet1").Cells(intRow, 2) = "Address"

'Now read the values from the text file…
Open strFileName For Input As #1 'Open the text file…
    Do Until EOF(1) 'process the entire file
        Line Input #1, strLineItem 'Each line of the text file is stored in strLineItem
        'Pass each line of the text file to a function and parse into separate columns
        strLineItem = Trim(strLineItem) 'Any preceding or trailing spaces are removed.
        'The Breakstring function (shown below) is used to break each like item at the 'tab. This is an ‘EXTREMELY useful function, and can be altered to meet your 'individual needs.
        LineResult = BreakString(strLineItem, vbTab)
        If LineResult(1) <> vbNullString Then 'If there is data in the line coming from the 'text file, then it is entered onto the worksheet.
            intRow = intRow + 1 'Increment the row count
            Worksheets("Sheet1").Cells(intRow, 1) = LineResult(1) 'Write the row value
            Worksheets("Sheet1").Cells(intRow, 2) = LineResult(2)
        End if
    Loop
Close #1
Exit Sub 'This need to be here, otherwise the error handling code is going to run.
errHandler: 'This is the errHandler label
Close #1 'to avoid error 55 (File already open)

MsgBox Err.Number & vbCrLf & Err.Description 'Notify the user of the error…

End Sub

Function BreakString(SomeText, Delimiter) As Variant

'Send an array back as a result to another variant in the calling proc

Dim intStart As Integer

Dim intNextDelim As Integer

Dim strSplitText() As String

Dim intCounter As Integer

If Right(SomeText, 1) <> Delimiter Then 'make sure the last column gets counted
    SomeText = SomeText & Delimiter 'add the delimiter to the last column
End If

intStart = 1 'the starting position always changes, so it is a variable

intNextDelim = InStr(1, SomeText, Delimiter)

Do While intNextDelim >= 1 'stop when 0 (there are no more delimiters)
    intCounter = intCounter + 1

    ReDim Preserve strSplitText(intCounter)

    strSplitText(intCounter) = Mid(SomeText, intStart, intNextDelim - intStart) 'store value in an array

    intStart = intNextDelim + 1 'move the start position up one

    intNextDelim = InStr(intStart, SomeText, Delimiter) 'find the next delimiter

Loop

BreakString = strSplitText 'return the array to the variant

End Function
**VBA Read Excel Files**

Sometimes you need to read one Excel sheet and write the values to another Excel sheet. This code takes the values from one Excel sheet and writes them to another sheet.

You can modify this code to select from the worksheet based on some criteria and then write the value that you need (based on that criteria) to another.

**Basic**

```vba
Public Sub ReadExcelFile()
    Dim strCellValue As String
    'Read the cell and assign the cell value to a variable...
    strCellValue = Worksheets("Sheet1").Range("A1")
    'Write the cell value to a destination cell
    Worksheets("Sheet2").Range("B1") = strCellValue
End Sub
```
**VBA Error Handling**

Proper error handling helps to avoid user confusion when an error is encountered. It gives the user some sort of direction and can help you, the programmer, fix the troubled area.

**Basic**

Public Sub ErrorSample()
    Dim intNumber As Integer

    On Error GoTo errHandler 'If there is an error go to the “errHandler” label…
    intNumber = 3 / 0 'This causes an error…
    Exit Sub 'This need to be here, otherwise the error handling code is going to run.

errHandler: 'This is the errHandler label
    MsgBox Err.Number & Err.Description 'Notify the user of the error number and error description…

End Sub
Putting it all together

The functions and procedures listed in this report are the most commonly used.

Expand upon the examples listed here, but you can use these as a springboard.

Learning the items outlined in this report will help you:

- Learn VBA over the weekend!
- Focus on what you really need to know!
- Gain a new respect from your peers!
- Help you get that deserved raise at work!
- Helps you relax this weekend instead of stressing how do to do your task on Monday!

Thanks for reading this. Feel free to send me your questions and comments at erik [at] vbastring [com].

PS:

I recommend that you get the videos which demonstrate the concepts contained in this manual. Go to http://yourexcelvbatraining.com/cc_videos for details.