Learning from testing

Tips for technical communicators from testers

Communicator

The Institute of Scientific and Technical Communicators
Spring 2016

Reduce your long-term maintenance workload

Learn how Green Technology is relevant to you

DocOps and the technical communicator

The impact of Augmented Reality for us

Creating VBA loops: part 1

To save repeating code, Mike Mee looks at looping the loop.

Introduction

This will be a two-part article as the VBA (Visual Basic for Applications) process/function is an expansive subject to explain. Over both articles, I will be covering the various types of loop functionality available within VBA. Loops are the time savers within VBA that allow you to do *things*, one after the other, with ease.

There are a number of ways to perform a loop in VBA, and they are:

1. `For … Next`
   Looping through a series of numbers (forwards or in reverse) sequentially or in steps.
2. `Do … Loop Until`
   Looping through a series of statements continually until a particular value/result occurs.
3. `While … Wend`
   Looping through a series of statements until a particular value/result happens, or has happened.
4. `For Each x In y`
   Looping through a series of objects in your Word document, for example documents, tables, comments or images.

Depending on the type of loop, you can test for a particular condition to prevent a loop from running indefinitely. If you do find yourself watching an infinite loop, the only way to break out of it is by using the CTRL and Break key. Generally, this does not apply to `For … Next` loops as you are telling VBA the start and end point of the loop.

This first article will cover the `For … Next` loop, and the second will cover the remaining types.

Pseudo-code and VBA code

Within each loop type definition, I have included some pseudo-code. This explains how each type of loop works. After the pseudo-code will be some examples that you can type into the VBA Editor.

I suggest keeping all of the loop VBA examples in one document, so that you can return to each one, edit/tweak and then test. Before you create your macro, you need some test text to use with it. If you go into the main Word document display and type[1]:

```
= lorem(5,10)
```

Press ENTER on the keyboard and five paragraphs containing ten sentences of pseudo Latin sample text are created.

For . . . Next loops

This type of loop is frequently used in most forms of BASIC, not just VBA. It will let you run through a series of values from `x` to `y`.

Pseudo-code

```
For counter = start to end
[Code statements/functions]
[Test For Exit]
[Additional Code statements/functions]
Next [counter]
```

The above pseudo-code shows the general layout of a `For … Next` loop: You need a variable that will be the counter for every iteration through the loop, a starting value and an end value. Within the loop is the VBA code that you want to perform during the loop.

If necessary, you can add a test, which exits the loop, using the `Exit` command, before it reaches the value at the end of the loop.

The `Next` command increases the counter by one and restarts the loop.

In the following example, I have set `intNumbers` as the counter variable, my start is set to `1` and my end is set to `100`. Display the Intermediate Window, type the code into a new module and run it to see what happens.

```
Sub ForNextSimpleExample()
Dim intNumbers As Integer
For intNumbers = 1 To 100
      Debug.Print intNumbers
Next intNumbers
End Sub
```

It is not very exciting code, as it is just going to print these numbers to the Immediate Window within the VBA Editor. However, it does show you the basics of how to create a simple VBA loop.

You could add the `Exit` command before the `Debug.Print` command with a simple test that checks if the loop is above `80` and abandons the loop if it is true.

```
Sub ForNextSimpleExample()
Dim intNumbers As Integer
For intNumbers = 1 To 100
      If intNumbers > 80 Then Exit For
      Debug.Print intNumbers
Next intNumbers
End Sub
```

The `Exit` command has to be placed before the `Debug.Print`, otherwise the loop would print the number `81` in the Immediate Window.

[1] Microsoft (2011) How to insert sample text into a document in Word https://support.microsoft.com/en-us/kb/212251 (accessed January 2016)

For . . . Next loops (in reverse)

These are the same as the previous `For … Next` loops, but you go through the values in reverse order.

Pseudo-code

```
For counter = end to start Step -<Step>
[Code statements/functions]
[Test For Exit]
[Additional Code statements/functions]
Next [counter]
```

The pseudo-code above is not much different from the loop going forwards. All I have added is the `Step` command to tell VBA that I want to force the loop to go backwards. For example, going backwards in a loop in steps of one, you enter `-1` as the value. Here is an example showing a `For … Next` loop in reverse.

```
Sub ForNextSimpleReverseExample()
Dim intNumbers As Integer
For intNumbers = 100 To 0 Step -5
      Debug.Print intNumbers
Next intNumbers
End Sub
```

The example above will start by displaying `100` and then every fifth number from `95` down to `0` in the Immediate Window.

As with the previous `For … Next` loop example, you can add in an exit clause using the same syntax and it will jump out at say, `50`, or `45`, depending on where you place the command.

You can also use the `Step` command for any kind of leaps within a loop: every second number, every third word or every fourth paragraph. It is up to you where and when you use it.

Referring to Word objects

We have been using numbers in loops and doing simple tasks with them. As this column is about programming Word VBA, we had better start using loops with Word objects. This will then introduce you to the other types of loop as well.

When referring to Word objects in your VBA code, you will find that the Object Browser (via the F2 key in the VBA Editor) will come in handy.

All of the Word objects that you will (mostly) interact with are stored in the `ActiveDocument` collection. This can be a table, a section, any comment or an image.

As a tip for optimising your VBA code, when you are referring to the `ActiveDocument` collection several times in a single macro, you can use the `With` command to reduce the amount of VBA code you have to type. This also has the added benefit of speeding up your code.

In the examples that follow, I will include the `With` command a lot, partly for optimisation and partly so that it can fit the code into the article column widths.

Real world examples

I have included two examples where a `For … Next` loop is used to traverse through a collection of Word objects and do something with them.

The first will reset any paper sizes set up as Letter, and fix them to be A4. The second will loop through and delete any images in the document.

Reset to A4

For example, you receive a document where each section is in Letter format, but you need to change it to A4. The following macro will go through all of the sections in the document and change each one from Letter to A4.

```
Sub LetterToA4()
Dim intSections As Integer
With ActiveDocument
   For intSections = 1 To .Sections.Count
   With .Sections(intSections).PageSetup
   If .PaperSize = wdPaperLetter Then
      .PaperSize = wdPaperA4
   End If
   End With
   Next intSections
End With
Msgbox intSections & " sections were
examined."
End Sub
```

There are two `With` commands in the above example. The first `With` covers the `ActiveDocument` and the second covers the `PageSetup` value within each section. This has saved you having to enter this long command - `ActiveDocument.Sections.PageSetup.PaperSize` - several times in your code. The total number of sections examined appear in a message box after the loop is finished. For example, if you have 3 sections in your document, the message box will show 4 as this is the number the loop reached when it exited the loop.

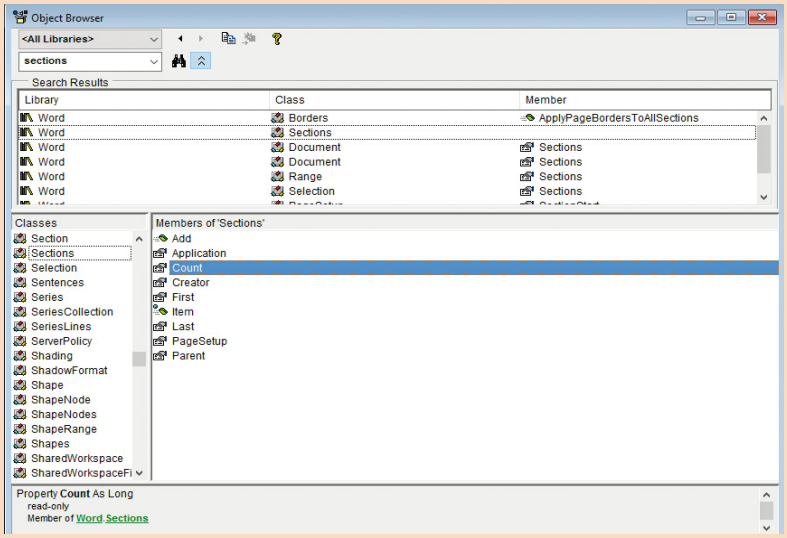

Figure 1. Object Browser

Deleting all pictures

This example uses a `For … Next` loop, but in reverse. This method is handy for tweaking or removing particular objects from your document, but without causing Word to keep re-arranging page numbers etc, as could happen if you used a loop going forwards.

```
Sub DeleteAllImagesReverse()
Dim lngImg As Long
With ActiveDocument
For lngImg = .InlineShapes.Count To 1
Step -1
      .InlineShapes(lngImg).Delete
Next lngImg
End With
End Sub
```

The variable `lngImg` is used to count backwards from the total number of images found back to `1`. The reason for using a Long instead of an Integer is that most Word objects that have an associated `Count` command will return a Long value.

Re-using the code

The same code could be re-used with a few changes to remove all comments, tables and other objects from a Word document.

However, when you are coding these kinds of routine, always ensure it is on a backup of a document and play around carefully. The last thing you want to find out is that your latest magnum opus is suddenly missing a lot of content due to running a test macro out on it.

Also, as you will find out over time, there are commands available in VBA already that do the work for you without having to use a loop, such as `ActiveDocument.DeleteAllComments`. Instead of creating a loop to go through and examine all of the comments in your document, you just have to issue a single command.

The end (of the loop)?

I have covered the first type of loop and the follow-up article will cover the remainder. Whilst you are waiting for the next issue, you can have a play with the `For … Next` loop and see how it can speed up some of the slower tasks in Word. Good luck!

Mike Mee MISTC is a technical author working at CDL in Stockport.
E: mike.mee@cdl.co.uk
T: @Mug_UK
W: http://mikestoolbox.weebly.com – my toolkit for Word 2007-2016 (the full source code is also available on the website).