Software-related articles

SCHEMA ST4, MadCap Central , Word VBA, DITA

Communicator

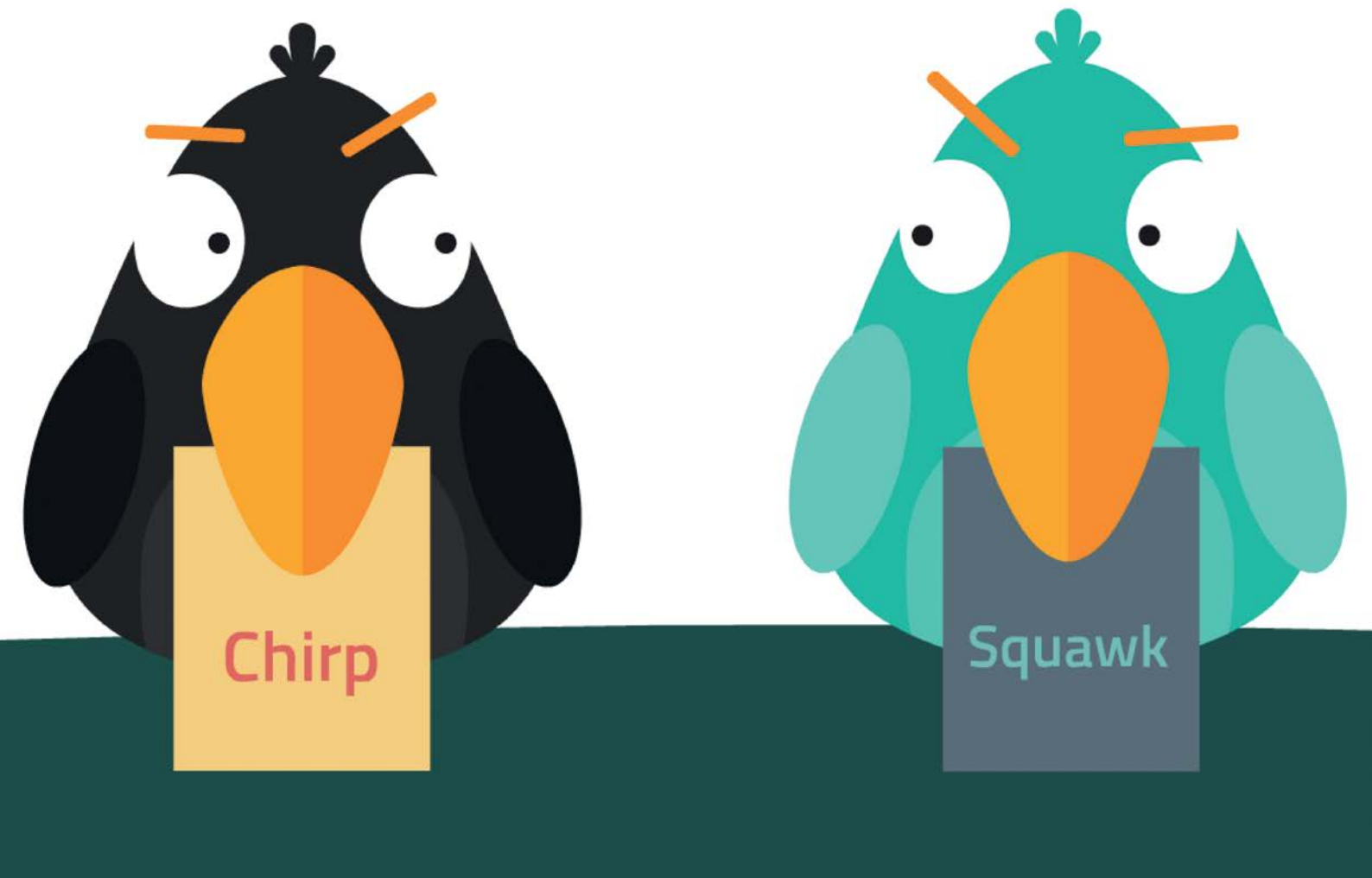
The Institute of Scientific and Technical Communicators
Spring 2017

Can you disconnect from work?

Learn all about sub-editing

Tips to improve your confidence

Develop your content strategy



Optimising your VBA code

Mike Mee explains how to improve your code to make it run faster.

Introduction

Welcome to the eighth article in my series about macro programming for Word.

This is a shorter article, compared to my normal ones, which will cover how to speed up your VBA code. The optimisations described are ones that I discovered while trying to reduce the size of my Word Toolbox add-in by a few kilobytes prior to its release in November 2016.

The IIF command

This simple command takes the `If Then Else` statement to a new level. It was one of the commands that I discovered and then set about using in my Word Toolbox.

Instead of writing the (usual) multiple lines of code for a simple `If Then` test, you just use a single line of code.

The `IIF` command is very similar to Excel's `If` functionality in that you include a test, then a result if the test is `True`, followed by another result if the test is `False`. All in a single line of code. The result of the test is stored in the initial variable that appears before the `IIF` command.

How it works

Take example 1 in Figure 1, which uses `If Then Else` and compare it with example 2 which uses the `IIF` command.

Whilst the one line of code is a lot longer (Figure 2), you can replace five lines (Figure 1) of VBA code, with one.

```
Dim strDiff1st As String
With ActiveDocument.Sections(1).PageSetup
  If .DifferentFirstPageHeaderFooter = True Then
     strDiff1st = "Yes"
  Else
     strDiff1st = "No"
  End If
MsgBox strDiff1st & " the first section in this document has
Different First Page Header or Footer switched on."
  End With
```

Example A showing five lines of code using If... Then... Else

```
Dim strDiff1st As String
With ActiveDocument.Sections(1).PageSetup
  strDiff1st = IIF(.DifferentFirstPageHeaderFooter
  =True,"Yes","No")
MsgBox strDiff1st & " the first section in this document has
Different First Page Header or Footer switched on."
End With
```

Example B showing one line of code using the IIF command.

Figure 1. Examples showing the difference in coding using `If Then Else` and `IIF`

What IIF can't do

There are situations when the `IIF` command won't be able to shrink the number of lines of code.

If you have multiple operations that follow each other after checking a value, then these will require separate `IIF` commands.

Similarly, you can't use the `IIF` command to replace any `If Then ElseIf Else End If` checks.

It's not for everyone

There are times, however, when you might prefer the original layout of the code; maybe when you are just starting out with a new function and you want to make sure you understand it all properly. Write it out in 'long form' first and then optimise it if required.

There are also those who prefer the extended way that the code is displayed because to their eyes, it is easier to read. In this case, it is up to your own preference.

Toggling True/False values

This optimisation trick only works on objects that can only hold a value of `True` or `False`.

However, this does cover a lot of Word's built-in options, any font style characteristics (such as bold, italic) or those options which are linked to each open document, such as the enforcing of styles embedded in a template.

To do this, we need to use the `Not` function. This function inverts the value of the Boolean variable passed to it. So if the value was `True`, it becomes `False` and vice-versa.

In the example below, the code is toggling the enforcement of styles feature.

How it works

Take this example where the `EnforceStyle` status of a document is toggled:

```
With ActiveDocument
If .EnforceStyle = True Then
   .EnforceStyle = False
Else
  .EnforceStyle = True
End If
End With
```

Now look at it using the `Not` command instead.

```
With ActiveDocument
   .EnforceStyle = Not .EnforceStyle
End With
```

You can also convert it into a single line by removing the `With ActiveDocument / End With` surround and referring to it as:

```
ActiveDocument.EnforceStyle = Not
ActiveDocument.EnforceStyle
```

Whichever of the two variants you prefer to use, you have converted seven separate lines of VBA code into either three lines, or a single one!

Merging subroutines

This is the final optimisation suggestion for this article.

In theory, as your VBA code grows and grows, you might find yourself copying and pasting an existing routine, as it closely matches what you need, but then tweaking it slightly.

I found this happened quite a bit in my Word Toolbox and, in the latest version, I found a way to remove the numerous duplicate lines of

```
' These are the new calling routines

Private Sub TableResizeToggle()
  TableToggle ("Automatic Resize")
End Sub

Private Sub TableWrapAround()
  TableToggle ("Wrap Around")
End Sub

Private Sub AllowBreakAcross()
  TableToggle ("Allow Rows Break Across")
End Sub

Private Sub BordersOnOff()
  TableToggle ("Borders")
End Sub

' This is the replacement routine that toggles certain table settings

Private Sub TableToggle(strTableToggle)
On Error GoTo TableResize_ErrorHandler

' Make sure parameter has a value inside it
If strTableToggle = vbNullString Then
     Exit Sub
Else
    'Loop through *all* tables
    For lngUpdateTable = 0 To ActiveDocument.Tables.Count - 1
      With ActiveDocument.Tables(lngUpdateTable)

        Select Case strTableToggle
        Case "Automatic Resize"
          .AllowAutoFit = Not .AllowAutoFit

        Case "Wrap Around"
          .Rows.WrapAroundText = Not .Rows.WrapAroundText

        Case "Allow Rows Break Across"
          .Rows.AllowBreakAcrossPages = Not .Rows.AllowBreakAcrossPages

        Case "Borders"
          .Borders.Enable = Not .Borders.Enable

        ' Exit on error, just in case
        Case Else
          Exit Sub
        End Select
      End With

    Next lngUpdateTable
End If

End Sub
```

Figure 2. Merging subroutines

matching code, by merging several routines into a single routine. I then created a simple routine to call up this newly merged main routine. This happened in a few places.

When working with the multiple toggle functions I had created for the Table Inspector, I had (lazily) created the initial routine, then copied and pasted it a further three times. Each copy had one or two lines of code changed within each duplicate. Not my best effort!

The code in Figure 3 replaced the existing copying and pasting and helped me to remove (approximately) 70 lines of code.

Each original repeat of the code was replaced by a simple call to the main subroutine, which was passed a parameter, as a string, to refer to the table toggle functionality required.

The main routine examined the parameter and then applied the fix to all of the tables in the document.

To set this up within your own code library, you will need to create four new Quick Parts which point to the four macros at the start of the code.

Please be aware that after setting up each Quick Parts icon, clicking one will launch the appropriate calling routine and apply the fix to **all** of the tables in your document.

In the original code, which you can download from my website, you will see that this function is slightly different. This is because I am referring to tables in a document that had been selected from a list box on a form; these are two items that I have not covered in any of my articles as of yet.

Next article

The next article will cover the basic steps in the creation of a form, via the use of Quick Parts, so that you can (eventually) add a GUI (or UX in modern parlance) to front your VBA code.

Support

If you get stuck trying to recreate any of the subroutines covered in this article, you can contact me via email, Twitter or put a post on the ISTC forum.

Further reading

Mee M (2015) 'Variables and screen output/input' *Communicator*, Autumn 2015: 44-47

Mee M (2016) 'Creating VBA loops: part 1' *Communicator*, Spring 2016: 34-36

Mee M (2016) 'Creating VBA loops: part 2' *Communicator*, Summer 2016: 52-55

Mee M (2016) ' Document content and manipulation' *Communicator*, Autumn 2016: 55-58

Microsoft 'Declaring Variables' https://msdn.microsoft.com/en-us/library/office/gg264241.aspx?f=255&MSPPError=-2147217396 (accessed April 2016)

Mike Mee MISTC is a technical author with 14 years' experience in the software industry.
E: mugukmail@gmail.com
T: @Mug_UK
W: www.mikestoolbox.co.uk – my toolkit for Word 2007-2016 (the full source code is also available on the website).