Learn, network and share

Read about this year's TCUK conference

Communicator

The Institute of Scientific and Technical Communicators
Winter 2015

Localise content through web design

The world of analytical engineering

Learn more about networks

Are technical communicators grumpy?

Checking VBA variable contents

If this, then do that. Or else do that. And more. By Mike Mee.

Introduction

Welcome to the third in my series of articles covering the use of VBA within Word.

This article will be covering the code that checks the value of a variable that you have created and then decides what will happen next.

You could also use the same method to check the value of a Word variable, such as the colour of a piece of text.

Modern methods

VBA has two methods for examining variables and this article covers their use.

1. `IF .. THEN .. ELSE .. END IF`
2. `SELECT .. CASE .. END SELECT`

IF .. THEN .. END IF

This is the simpler of the two types to use and will probably cover the majority of your variable checks.

At the simplest level, you use an `IF .. THEN` construct to examine the contents of a variable and then do *something*.

A simple IF.. THEN example

If you need to ask the end user their age, you might want to display a message on-screen if it is under 18. This simple example will do just that.

```
Sub AgedOver18()
Dim intAge AsInteger
intAge = InputBox("Please type in your age
in years")
If intAge < 18 Then
  MsgBox "You are too young to be a
technical communicator!"
End If
End Sub
```

The program displays an `InputBox` with a simple question. The user enters their age and the `IF` statement examines their age. If they input an age value that is less than 18, then a `MsgBox` is displayed with the error.

You will notice there is no checking for anyone who enters their age if it is over 18. That requires the use of the `ELSE` command and it is covered in the next section.

Introducing ELSE

When you need to do more than one thing based on the result of an `IF` check, you will need to use the `ELSE` command.

The premise of this type of conditional check is one where you need to examine the contents of a variable and if it matches `'Condition A'` then do *something*, but if it matches 'Condition B', do *something else*.

An enhancement using ELSE

In the first age check example, we did not check if the age entered was anything but under 18.

Using the `ELSE` command, we can add in some extra code that will display a message if the user is over 18.

```
Sub AgedOver18()
Dim intAge As Integer
intAge = InputBox("Please type in your age
in years")
If intAge < 18 Then
  MsgBox "You are too young to be a
technical communicator!"
Else
  MsgBox "You are a technical communicator."
End If
End Sub
```

The use of `ELSE` allows the code to display the correct message depending on the age value entered.

Using Booleans

In the previous article, I introduced you to the Boolean variable type. Booleans are simple variable types used to check if something is `TRUE` or `FALSE`. The code used a Boolean to check if the user was a valid ISTC Member (or not).

At the start of the code, the variable `bolISTCMember` is set to `TRUE`. The remainder of the code asks a number of questions and, if any of the answers given were not valid, the value was set to `FALSE`.

At the end of the routine are the following lines:

```
If bolISTCMember = True Then
  MsgBox "Welcome to ISTC, " & strName &
"!", vbInformation
Else
  MsgBox "Either your name or your membership
number was not recognised.", vbCritical
End If
```

The `ELSE` command ensured that if the Boolean variable resulted in a `TRUE` result, the person was a genuine ISTC member. If it was a `FALSE` result, they were an 'imposter'.

A more extensive IF example

In this extended example, we are going to display a particular message on screen (using `MsgBox`) based on the answer given for that person's age.

```
Sub AgeCheck()
Dim intAge As Integer
intAge = InputBox("Please type in your age
in years")
If intAge < 18 Then
  MsgBox "You are too young to be a
technical communicator."
End If
If intAge >= 18 And intAge <= 30 Then
  MsgBox "In your prime."
End If
If intAge > 30 And intAge < 65 Then
  MsgBox "Beavering away."
End If
If intAge >= 65 Then
  MsgBox "Enjoying retirement?"
End If
End Sub
```

You could merge this into the example code in the previous issue; the code will be able to ask the age of the ISTC member.

There is also ELSEIF

There is an additional statement that you can add into your `IF .. THEN .. ELSE` constructs.

This command is `ELSEIF` and while it can be nested in a long `IF .. THEN .. ELSE` construct, it can make your code look long-winded (aka 'spaghetti-code'). This is especially so when compared to the `SELECT .. CASE` construct, covered in the next section.

```
Sub AgeCheck_ElseIf()
Dim intAge As Integer
intAge = InputBox("Please type in your age
in years")
If intAge < 18 Then
  MsgBox "You are too young to be a
technical communicator."
ElseIf intAge >= 18 And intAge <= 30 Then
  MsgBox "In your prime."
ElseIf intAge > 30 And intAge < 65 Then
  MsgBox "Beavering away."
ElseIf intAge >= 65 Then
  MsgBox "Enjoying retirement?"
End If
End Sub
```

In the tweaked code, all the `ELSEIF` command has done is remove almost all of the `END IF` statements.

However, if you were to add more checks to the same code, such as splitting the 18 to 30 check in the middle, then it can start to get messy.

It is your choice what to use. If you can read the `IF .. THEN .. ELSE` commands easier, then stick to that. VBA does not force you to use the `ELSEIF` command.

The old days

While writing this article I thought about the old days, and the pain of checking a variable in the 1980s, when I first started learning to program on my ZX Spectrum.

Those early coding days of my youth

The ZX Spectrum I was using had, according to the adverts at the time, a 'massive' 48K of RAM that I could use for my own BASIC programs. This was not strictly true, as you did not get the full 48K to play with. However, if you compared this to a stock VIC-20 with its paltry 3.5K of RAM, the 48K was sheer luxury.

These days, a 'blank document' created in Word can take up that much space before you have added any text to it. Such is the price of progress.

The variant of BASIC that came with my ZX Spectrum was quite limited. This was down to it having to fit inside the computer's memory – the bits that you could not touch.

When it came to checking the value of a variable, you were limited to asking:

```
If the value of x is y then do this.
```

This could lead to a lot of BASIC code where you would have to create two separate statements, one for the positive outcome and one for the negative outcome. If you were expecting many (possible) answers, then you had to be prepared to create a lot of separate `IF` statements.

Thankfully, those days are long gone and when you are using VBA, you now have better methods of examining your variables' values.

Although, it would have been bliss to have an ELSE command in the ZX Spectrum's BASIC interpreter!

Figure 1. `IF` statements in BASIC

SELECT .. CASE .. END SELECT

This second type of variable checking is better suited to longer lists of checks against a single variable.

You define the variable in which are interested via the `SELECT` command and then add a separate `CASE` statement for the each of checks you need. To close off the list, you add the `END SELECT` command.

To show the difference between `SELECT .. CASE` and `IF .. ENDIF`, the following code performs the same function as the previous example but has been re-written using `SELECT .. CASE` statements.

```
Sub AgeCheck()
Dim intAge As Integer
intAge = InputBox("Please type in your age
in years")
Select Case intAge
  Case Is <18
     Msgbox "You are too young to be a
     technical communicator."
  Case 18 To 30
     Msgbox "In your prime."
  Case 31 To 64
     Msgbox "Beavering away."
  Case Is >= 65
     Msgbox "Enjoying retirement?"
End Select
End Sub
```

You might notice how 'cleaner' a series of SELECT .. CASE commands looks compared to a multitude of IF .. THEN .. ELSE commands.

The way you define a range of values, to me at least, seems neater. You just separate the lower and higher values with a TO command.

Examining Word variables

You can use either of the two types of construct to examine the values of Word's built-in variables.

In its simplest form, you can examine a Word variable to perform a quick check of the current document's read/write status. For example:

```
If ActiveDocument.ReadOnly = True Then
  MsgBox "This document is read-only!"
End If
```

In the example above, I display a message, but you might want to expand on this and maybe exit the current function of your VBA code if the file is read-only.

Likewise with SELECT .. CASE commands. In my Word Toolbox, I check what the language type is for the template attached to the document with this code:

```
Sub CheckTemplateLanguage()
Dim txtLang As String
With ActiveDocument.AttachedTemplate
  Select Case .LanguageID
     Case 2057, 1033
        txtLang = Languages(.LanguageID)
     Case wdUndefined
        txtLang = "Undefined/Multiple"
     Case Else
        txtLang = "Other (not UK or US)"
   End Select
   Msgbox "This template uses: " & txtLang
End With
End Sub
```

There is no need to worry about the With and End With statements that I've used in the code, as I will be covering them, and other types of code looping, in the next article.

The SELECT statement investigates the value of the Word variable LanguageID and, based on what is there, stores the language in the string txtLang. The SELECT command also checks if the document is using multiple languages or something other than UK or US.

After the routine has checked the language, it displays one of three messages based on what it found.

In case you are not sure what the values are in the first Case statement, these are what Windows uses to store the country code. 'English (UK)' is stored as 2057 and the 'English (USA)' is 1033.

I did not include the full list of languages available, as there are more than 200 in Word 2013!

It is your choice in the end

As you progress through creating your own code, you will come across situations where a simple IF .. THEN .. ELSE will work better than a list of SELECT .. CASE commands.

Try using both types of command in the same scenario and work out which is easiest to use. The most important thing to consider is that, when you return to the code in a few months, "Which will be the easiest to update?". C

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).