

Study at the University of Limerick

Learn about a virtual technical communication course



Communicator

The Institute of Scientific and Technical Communicators
Autumn 2015

**Viable options for
replacing QWERTY**

**Have you thought about
working remotely?**

**Do women make better
technical communicators?**

**RoboHelp 2015 and
PerfectIt product reviews**



Variables and screen output/input

How to use variables and message boxes by [Mike Mee](#).

In my first article, I covered the recording and editing of macros. In this article, I will show you the next steps, which is the use of variables (and constants), followed by displaying information to, and getting it from, the end user.

What are variables and constants?

Variables are used in programming languages to store values. They are areas of memory that, in this case, VBA, can use to store values, such as a user's name, their age, or their ISTC Membership number.

In VBA, there are two types available: variables and constants. Variables, as their name suggests, can have their contents changed, whereas constants have a value assigned, and the code cannot change it.

Defining variables

Each variable is defined – that is, given an area of memory to exist – by the use of the `Dim` command in VBA. The name of the variable is inserted after the `Dim` command, like this example:

```
Dim [VariableName]
```

In other languages based around BASIC, this is normally all you would need to type. The variable will exist in memory and you can store whatever you like inside it.

However, VBA can define variables to hold particular types of data only. Without adding a data type (as per the above example), your variable has become a *variant* data type because it can store anything you want inside it.

You can define what type of data you want

to store in this new variable, by defining its data type. The data type is specified after the new variable's name, as shown in this example:

```
Dim [VariableName] As [TypeOfVariable]
```

VBA provides a number of data types you can use for variables, see Table 1.

Variable names must start with a letter. They can be as long as 40 characters but can contain only letters, numerals, and the underscore (`_`). However, VBA-reserved words (such as function names or statements) are not available to use as variable names. For example, you cannot use `Dim` as a variable name, because it already exists as a function.

Examples of variables

Using some of the above data types in Table 1, here are some variable name examples:

```
Dim strName As String
Dim intMembershipNum As Integer
Dim bolISTCMember As Boolean
```

These variables now exist in memory and VBA knows what it can store inside them. I will be using these variables in the input/output section of this article later.

Defining your constants

Defining a constant is similar to the method for defining variables, but you must include the value that the constant will permanently hold.

Naming conventions

Out of habit, I define my variables along the lines of the Leszynski naming convention which makes it easier to read and/or debug your code. I prefix each variable name with a three-letter code, which matches the type of variable.

Whenever I see `strName` in my code, I know that it is a string variable and `intMembershipNumber` will only allow integers.

However, if you prefer to use your own naming scheme, go with that, although I would try to avoid using single letters to define a variable name. Trying to work out if the variable named 'a' represents a string, an integer or a Boolean would be painful in larger macros. http://en.wikipedia.org/wiki/Leszynski_naming_convention#The_LNC_Tags_for_VBA_Variables

Figure 1. Tips for naming variables

Table 1. Types of variables

| Variable Type | What can be stored in it |
|---------------|--|
| Boolean | True or False values. |
| Byte | 0 to 255. |
| Integer | 32,768 to +32,768. |
| Long | Values can be from approximately -2 billion to +2 billion. |
| Currency | Decimal numbers with as many as 19 digits. |
| Single | Single-precision, floating-point numbers. |
| Double | Double-precision, floating-point numbers. |
| Date | A date or time value. |
| String | Text |
| Object | Any object, such as a Word document or a window. |
| Variant | A generic number or string. |

Note: Single or Double variables are more useful in Excel than Word.

For example, you need to store the value for the version number of your program. You know that this value is not going to change, not until you release a new version, so you can define it as such:

```
Const strVersionNum As String = "Alpha"
```

A constant uses `Const` whereas a variable uses `Dim`. The above example will **never** change whilst the code is running. You cannot alter it on the fly, so wherever you use it in your code, it will always contain the word 'Alpha'.

The same rules apply to defining constants as they do for variables. If you try to define a constant as a string, you will not be able to store an integer in it. As an example, here is an **incorrect** definition of a constant:

```
Const strVersionNum As Integer =
"Alpha"
```

The VBA Editor will complain as soon as you press Enter to submit that line of code because you are trying to store a string inside an integer.

Word's built-in constants

Within VBA, there are constants that you can use that are built-in. They generally help to make your code readable.

To access all of the built-in constants, you have to use the Object Browser function whilst you are in the VBA Editor. Either press F2 or click on **View > Object Browser**.

The Object Browser will give you the background on any of the constants that are available for you to use or examine. However, it does not just cover the constants that are built-in. Explaining what it will show you is out of the range of this article, so I will upload a more expansive view onto the ISTC website.

The Object Browser shows you the direct value of the constant in the small window underneath. In the example above, I have selected the built-in constant `wdYellow`. This is because, in the last article, you changed the colour of the text to yellow. You can see that `wdYellow` is a member of the `wdColorIndex` class, which also holds all of the other standard colours available. The colours and their associated constants appear in the larger window below.

You could, if you wanted to, return to your original macro, and change the code so that `wdYellow` is now `wdPink`, see Figure 3. Re-run it to change the text from yellow to pink.

Switching on Option Explicit

Before I move on to screen input and output, one last tip for tidy macro coding is to switch on VBA's Option Explicit option.

This allows VBA to ensure that you define every variable prior to its use. If you refer to

the variable `strName` before VBA has found a matching `Dim` definition for it, it will stop the macro at that point with an error message.

This error will also crop up if you have defined the `strName` within a module, but then tried to use it elsewhere. See the previous section: 'Where to set up your variables'.

You can switch on Option Explicit at module level, by typing in `Option Explicit` at the top of every module you use.

Alternatively, you can switch it on at project level so that VBA automatically adds it to each module by selecting the **Require Variable Declaration** option which is under **Tools > Options > Editor** within the VBA Editor.

Screen output and input

As I will not be covering the use of your own custom designed forms for some time to come in these articles, the second half of this issue's article will cover message and input boxes.

These methods are the easiest way to either display a message to the user or request some input from them - hence their names.

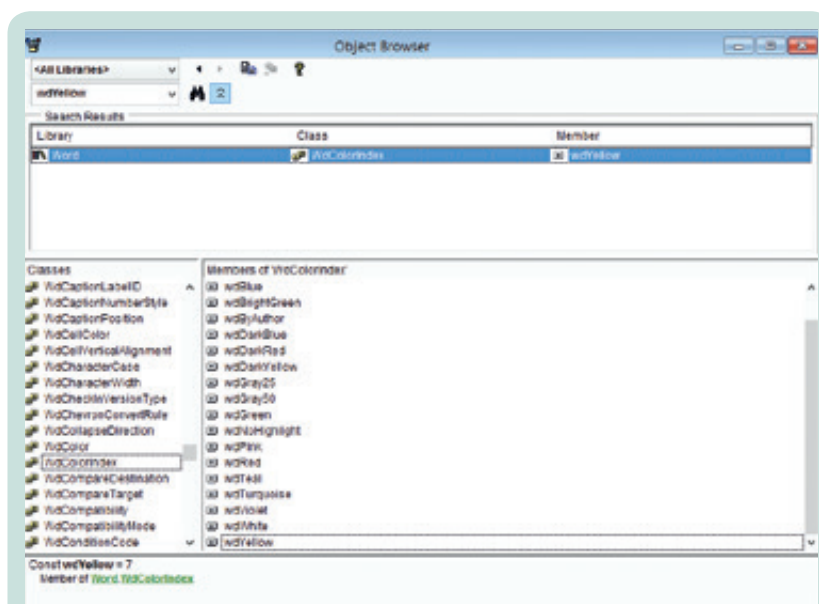


Figure 2. Object Browser

```
Sub MyFirstMacro()
    '
    ' MyFirstMacro Macro
    '

    Selection.WholeStory
    Selection.ParagraphFormat.Alignment =
        wdAlignParagraphRight
    Selection.Font.Size = 20
    Selection.Range.HighlightColorIndex = wdPink
    Selection.Font.Bold = True
End Sub
```

Figure 3. Amended code example from *Communicator Summer 2015*

Message boxes

First, I will cover message boxes. These take the form of a simple dialog that appears in the centre of your screen. The user reads the text inside the message box and then clicks one of the button(s) available.

The simplest example is one that just displays a message and the end user clicks the **OK** button.

```
Sub Display_Message()  
    MsgBox "Hello World!"  
End Sub
```

The above example will display the message within a message box dialog. The user cannot remove this dialog (known as a modal dialog) without clicking on the OK button that appears.

This could be useful, for example, in those situations when your macro has completed and you need to alert the user that the process has finished. Alternatively, you could use it as part of your debugging process by adding message boxes before and after a function to show the contents of variables used.





You can change the type of icon displayed inside the message box, by adding an additional parameter after your text message:

```
MsgBox "Hello World!", vbExclamation
```

In this example, you see the same 'Hello World' message on-screen, but instead the icon is now an exclamation mark.

There are several built-in constants in VBA that you can use, along with your message box, see Table 2.

Table 2. Built-in constants

| Constant | Icon |
|---------------|---|
| vbCritical |  |
| vbQuestion |  |
| vbExclamation |  |
| vbInformation |  |

Input boxes

The input box is similar to the message box in how it appears on screen, but you can ask the user to enter some information. The result is stored in a variable, so that you can examine the content afterwards.

```
Sub GetName()  
    Dim strName As String  
    strName = InputBox ("Enter your name")  
    MsgBox "You entered: " & strName  
End Sub
```

In the above example, I created a variable called `strName`, and then displayed an input box asking the user to 'Enter your name'. As soon as the user presses **Enter**, or clicks the **OK** button, whatever they have typed into the input box is now stored in `strName`. I can then display what the user entered via a message box.

Simple variable checking

Note: This topic will be covered in more detail in the next article. For now, I will cover some of the simpler functionality so that you can quickly test a response received from a message box or an input box.

VBA, as with all variants of the BASIC language, has the ability to examine the values of variables and constants. This can be done via the **If** and **Then** commands.

Using these commands you can test the contents of a variable and **If** `x` is a particular value, **Then** do something.

For example, you might want to check that your end user has entered something into `strName` after the `InputBox` asked them a question.

```
If strName = "" Then  
    MsgBox "Nothing was entered."  
End If
```

The code above is checking that the variable `strName` has nothing in it at all. This could be due to the end user either typing nothing at the prompt and clicking **Enter**, or clicking on the **Cancel** button.

Similarly, with message boxes, you can test the value of the button clicked. This will also be covered in the next article.

The end result

To end this article, the code example in Figure 4 will show how you can use variables, constants, message and input boxes. I have added a bit of an ISTC branding to it. **C**

Reference

Mee M (2015) 'Enhancing Word with VBA macros' *Communicator*, Summer 2015: 20–23



Mike Mee MISTC is a technical author working at CDL in Stockport.

T: @Mug_UK

E: mike.mee@cdl.co.uk

W: <http://mikestoolbox.weebly.com>

– my toolkit for Word 2007–2013

(the full source code is also available on the website).

```

Sub GetMemberDetails()

Dim strName As String
Dim intMembershipNum As Integer
Dim bolISTCMember As Boolean

' First we set up the Boolean to determine the validity of the ISTC member
bolISTCMember = True

' Display the welcome message
MsgBox "Welcome to ISTC. Please enter your name and membership number to continue.", vbInformation

' Ask their name
strName = InputBox("Enter your name")

' If nothing is entered, then we must assume they're not genuine ISTC members
If strName = "" Then
    MsgBox "No name was entered.", vbCritical
    bolISTCMember = False
End If

' Ask for their ISTC Membership number
intMembershipNum = InputBox("Enter your ISTC Membership number")

' If nothing is entered, then we must assume they're not genuine ISTC members
If intMembershipNum = "" Then
    MsgBox "No ISTC Membership number was entered.", vbCritical
    bolISTCMember = False
End If

' If they've entered both sets of details, we assume they're genuine.
If bolISTCMember = True Then
    MsgBox "Welcome to ISTC, " & strName & "!", vbInformation
Else
    MsgBox "Either your name or your membership number were not recognised.", vbCritical
End If

End Sub

```

Figure 4. A msgbox, inputbox and variable example.

“We have been very happy with the service provided by 3di. They have the know-how, the relevant experience and the project managers to deliver our projects and adhere to our high standards,,

Joseph Nosbuesh, Head of Documentation Management, Roche Diagnostics



Complexity made clear

Expert documentation & localization services

Technical Communication

- supply of technical authors
- managed outsourced teams
- information & document design
- modular writing training
- tools & process strategy
- software usability testing

Localization & Translation

- software products & online help
- website & e-learning content
- interactive & audio content
- technical & compliance documents
- multi-lingual translation
- scalable localization testing

☎ 01483 211533

✉ contact@3di-info.com

🌐 www.3di-info.com



See what else we offer...

If you enjoyed this article, visit our website to see what else we do.

The Institute of Scientific and Technical Communicators is the largest UK body representing information development professionals, serving both our members and the wider technical communication community.

What the ISTC offers



Professional development and recognition

Resources and opportunities to develop and diversify skills, stay up to date with trends and technologies, and get recognition for achievements.

Our CPD (Continuous Professional Development) framework enables you to provide evidence of your learning in all its forms, and our Awards programme gives you the opportunity to showcase excellent work.

Communicator



Communicator professional journal

Communicator is the ISTC's award-winning quarterly professional journal, covering the breadth of technical communications, offering in-depth articles, case studies, book and product reviews.

Now you've read a sample article, would you like to see more? The journal is free to our members and is also available on subscription.



ISTC Community

The ISTC offers opportunities to network, exchange expertise, and stay in touch with the UK technical communication industry – through a range of online groups, local events, and InfoPlus+ (our monthly newsletter).

You can find us on LinkedIn, Eventbrite, YouTube and Twitter (@ISTC_org).



Technical Communication UK

The ISTC hosts Technical Communication UK, the annual conference that aims to meet the needs of technical communicators, their managers and clients, from every corner of the industry.

Open to all, visit www.technicalcommunicationuk.com for the latest news.



ISTC Resources

The ISTC offers access to a range of resources, including our own books, various templates, articles summarising key technical communication issues and discounted British Standards publications.

