Centre for Learning and Academic Development (CLAD)
Technology Skills Development Team

MS Word and MS Excel: Macro Recorder and User-defined Functions

www.intranet.birmingham.ac.uk/itskills
MS Word and MS Excel: Macro Recorder and User-defined Functions (MWX2101)

Author: Sonia Lee Cooke
(Some of the functions in this course is based on a previous course written by Helen Gaffney)

Version: 1, October 2012

© 2012 The University of Birmingham

All rights reserved; no part of this publication may be photocopied, recorded or otherwise reproduced, stored in a retrieval system or transmitted in any form by any electrical or mechanical means without permission of the copyright holder.

Trademarks: Microsoft Windows is a registered trademark of Microsoft Corporation. All brand names and product names used in this handbook are trademarks, registered trademarks, or trade names of their respective holders.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>About the Workbook</td>
<td>1</td>
</tr>
<tr>
<td>Introduction</td>
<td>2</td>
</tr>
<tr>
<td>Visual Basic Programs (Macros)</td>
<td>2</td>
</tr>
<tr>
<td>The Macro Recorder</td>
<td>2</td>
</tr>
<tr>
<td>Examples of Macros in Word</td>
<td>3</td>
</tr>
<tr>
<td>Starting the Macro Recorder</td>
<td>3</td>
</tr>
<tr>
<td>Macro to Write Fractions in Text</td>
<td>3</td>
</tr>
<tr>
<td>Shortcut for Recording a New Macro</td>
<td>6</td>
</tr>
<tr>
<td>Macro to Number Headings in a Document in Which Heading Levels Have Been Assigned</td>
<td>7</td>
</tr>
<tr>
<td>Macro to Insert the Filename in Footer</td>
<td>7</td>
</tr>
<tr>
<td>Running Word Macros</td>
<td>9</td>
</tr>
<tr>
<td>Using the Developer tab</td>
<td>9</td>
</tr>
<tr>
<td>Assigning a Macro to a Shortcut Key</td>
<td>9</td>
</tr>
<tr>
<td>Edit a Macro in Word</td>
<td>10</td>
</tr>
<tr>
<td>Storing Macros Template in Word</td>
<td>11</td>
</tr>
<tr>
<td>Copying Macros From One Template to Another</td>
<td>11</td>
</tr>
<tr>
<td>Examples of Macros in Excel</td>
<td>12</td>
</tr>
<tr>
<td>Macro to Put a Bullet in a Cell in a Worksheet</td>
<td>12</td>
</tr>
<tr>
<td>Form Controls</td>
<td>13</td>
</tr>
<tr>
<td>Macro to Go to the Beginning of the Next Line When Entering Data in a Flat-File Database</td>
<td>14</td>
</tr>
<tr>
<td>The Visual Basic Editor</td>
<td>16</td>
</tr>
<tr>
<td>Open the Visual Basic Editor (VBE)</td>
<td>16</td>
</tr>
<tr>
<td>Code Window</td>
<td>17</td>
</tr>
<tr>
<td>Project Explorer</td>
<td>17</td>
</tr>
<tr>
<td>Properties Window</td>
<td>18</td>
</tr>
<tr>
<td>Customising the VBA Environment</td>
<td>19</td>
</tr>
<tr>
<td>Object Browser</td>
<td>20</td>
</tr>
<tr>
<td>Creating a Macro by Typing it into the Visual Basic Editor</td>
<td>21</td>
</tr>
<tr>
<td>Open the Visual Basic Editor (VBE)</td>
<td>21</td>
</tr>
<tr>
<td>Edit a Macro or User-Defined Function in Excel</td>
<td>22</td>
</tr>
<tr>
<td>Running Macros in Excel</td>
<td>23</td>
</tr>
<tr>
<td>Adding Macros to the Quick Access Toolbar</td>
<td>23</td>
</tr>
<tr>
<td>Attaching a Macro to a Button on a Worksheet</td>
<td>25</td>
</tr>
<tr>
<td>Assigning a Shortcut Key to a Macro in Excel</td>
<td>26</td>
</tr>
<tr>
<td>Creating a Macro to Resize a Graph</td>
<td>26</td>
</tr>
<tr>
<td>Storing Macros in Excel</td>
<td>27</td>
</tr>
<tr>
<td>Where to Store the Modules</td>
<td>28</td>
</tr>
<tr>
<td>The Personal.xlsp Workbook</td>
<td>28</td>
</tr>
<tr>
<td>Deleting Macros in Word and Excel</td>
<td>28</td>
</tr>
<tr>
<td>Deleting a Macro in Word</td>
<td>28</td>
</tr>
<tr>
<td>Deleting a Macro in Excel</td>
<td>29</td>
</tr>
<tr>
<td>User-Defined Functions in Excel</td>
<td>29</td>
</tr>
<tr>
<td>User-defined Function to Calculate the Area of a Triangle</td>
<td>30</td>
</tr>
<tr>
<td>Add a Description to Function Arguments Dialogue box</td>
<td>33</td>
</tr>
<tr>
<td>User-defined Functions to Convert Degrees Fahrenheit to Celsius and Vice Versa</td>
<td>34</td>
</tr>
<tr>
<td>Functions Available in VBA</td>
<td>35</td>
</tr>
</tbody>
</table>
USING FUNCTIONS IN OTHER WORKBOOKS ................................................................. 35
A USER-DEFINED WORKBOOK .............................................................................. 36
MACRO VIRUS PROTECTION .................................................................................. 36
REFERENCES ............................................................................................................. 37
About the workbook

The workbook is designed as a reference for you to use after the course has finished. The workbook is yours to take away with you so feel free to make any notes you need in the workbook itself.

The workbook is divided into sections with each section explaining about a particular feature of the program or how to do a particular task. Sections that take you through a particular procedure step-by-step look like this:

How to do something

- Do this first.
- Then do this.
- Then do this to finish.

There are also a number of text boxes to watch out for throughout the workbook. These will help you to get the most out of the course.

Tip

The thumbs-up symbol in the margin indicates a tip. These tips will help you work more effectively.

Danger!

The thumbs-down picture in the margin indicates common mistakes or pitfalls to be avoided.
Introduction

This course is designed to introduce the use of Macros in Word and Macros and User-Defined Functions in Excel (all Visual Basic Programs) to those who use Word and Excel quite a lot but have never got round to using them. In particular, the issues relating to the storing of modules – macros and user-defined functions – so that they are always available when needed – will be discussed.

The use of the Macro Recorder to record simple macros in Word and Excel will be covered. We will look at the code produced by the recorder and indicate how it can be modified to be more efficient. This is an effective way of not only learning some Visual Basic for Applications but also of learning how Word and Excel actually do the things they do; the instructions that Word and Excel give themselves!

Why might we want to record or write macros? Because if there are sequences of instructions that we repeat over and over again in exactly the same way every time we perform them it is advantageous to record the entire sequence so that Word or Excel can do them by carrying out a single instruction from us rather than several instructions. It might also be useful if a procedure is long and/or complicated and you don’t want to have to remember it every time.

Visual Basic Programs (Macros)

Excel 5 was the first of the Microsoft Office applications to use Visual Basic for Applications (VBA) as its programming language. The language is a subset of Visual Basic that now extends to Access, PowerPoint and Word. By using VBA, you can automate many tasks within the individual Office applications.

The Macro Recorder

The simplest method of producing a VBA program in both Word and Excel is to use the Macro Recorder. Although the work is done through the Macro Recorder, you will be creating a program in a high-level language. By looking at the code produced by the Recorder, you can become quite familiar with the language. And as already mentioned, it is also extremely useful in studying the ways in which Word and Excel perform actions.

The recorder is best used to automate fairly simple and straightforward tasks, such as those we are about to work through. There are a number of tasks that cannot be recorded; for example, conditional statements (If-Then-Else) in Excel, looping, displaying dialogue boxes, assigning values to variables.
Examples of macros in Word

We will start by recording a simple macro in Word; we will then look at the code and modify it to make it more efficient.

Starting the Macro Recorder

You can start the macro recorder by using the record macro button on the View tab; alternatively use the record button on the Developer tab.

To record macros do one of the following:

- Click on the View tab, in the Macros group click on the Macros button and select Record Macro…

- Click on the Developer tab, (to display the Developer tab if it’s not visible on the ribbon)
- Click on the File tab, click on the Options button (the Word Options dialogue box appears).
- Click on Customize Ribbon and on the right under Main tab, select Developer tab
- Click on the OK button to display the Developer tab on the ribbon.
- Click on the Developer tab and in the Code group, click on the Record Macro button.

Macro to write fractions in text

We will record a macro to help us write fractions in text, as opposed to fractions within a displayed equation, when you would probably use the Microsoft Equation facility.

The font sizes used in this example are probably suitable for documents in 10, 11, or 12 pt. You can always change the font sizes to suit your needs or taste.
Tip - Rehearse!
Before recording a macro using the Macro Recorder, it is as well to practise what you are going to do a couple of times before setting out to record it: first to make sure that it works (that the macro does what you want it to do), and secondly so that when you come to record you go through the instructions correctly in the most efficient way possible.

To create a macro to write fractions in text:

- Type a fraction in text; e.g. 5/12.
- Click on the Developer tab, in the Code group, click on Macros, the Macros dialogue box appears:

  Under **Macro name**: in the text box **Macro1** is the default name, type a name - e.g. **Fractions** – no spaces allowed in macro names (don’t worry at this stage about a **Keyboard** shortcut key).

  Click on the **OK** button. Notice that the cursor changes to and any thing you do will be recorded.
  
  The **Macro** button on the **Status Bar** at the bottom left of the screen gives you information about the macro.

  Press **Ctrl Left Arrow** three times, to move to the beginning of the fractions.

  Press **Shift Ctrl Right Arrow** once to select the numerator (which may be more than one digit).

  Click on the **Home** tab, in the **Font** group, click on the **Font** launcher to display the **Font** dialogue box.

  Change the font size to 8 or 9 pt and under **Effects** select **Superscript**.
- Click on the **Advanced** tab, under **Character Spacing** tab, click on the arrowhead to the right of **Position**: select **Raised** and raise the number by 1 pt.
- Click on the **OK** button to go back to the document.
- Press **Right Arrow** once to get past the first number.
- Press **Shift Right Arrow** to select the slash.
- Click on the **Insert** tab, in the **Symbols** group, click on **Symbol** and select **More Symbols...** (*the Symbols dialogue box appears*).
- Select the **Symbols** font, scroll down and select the forward slash shown below:

![Symbol](image.png)

- Click on the **Insert** button or **double-click** on the slash to insert it into the document.
- Click on the **Close** button to go back to the document.
- Press **Shift Ctrl Right Arrow** to select the second number – denominator.
- Click on the **Home** tab, in the **Font** group change the font size to make the denominator 8pt or 9pt depending on the size of your text.
- Press **Right Arrow** to cancel the highlight.
- Click on the **Home** tab, in the **Font** group change the font size back to 12 pt.
- **Stop** the recorder by clicking on the **Developer** tab, in the **Code** group, click on **Stop Recording**.
The Visual Basic code as created by the Macro Recorder to write fractions in text is shown below:

```vba
Sub fraction()
    ' fraction Macro
    Selection.MoveLeft Unit:=wdWord, Count:=3
    Selection.MoveRight Unit:=wdWord, Count:=1, Extend:=wdExtend
    With Selection.Font
        .Name = "Body"
        .Size = 9
        .Bold = False
        .Italic = False
        .Underline = wdUnderlineNone
        .UnderlineColor = wdColorAutomatic
        .StrikeThrough = False
        .DoubleStrikeThrough = False
        .Outline = False
        .Emboss = False
        .Shadow = False
        .Hidden = False
        .SmallCaps = False
        .AllCaps = False
        .Color = wdColorAutomatic
        .Engrave = False
        .Superscript = True
        .Subscript = False
        .Spacing = 0
        .Scaling = 100
        .Position = 1
        .Kerning = 0
        .Animation = wdAnimationNone
        .Ligatures = wdLigaturesNone
        .NumberSpacing = wdNumberSpacingDefault
        .NumberForm = wdNumberFormDefault
        .StylisticSet = wdStylisticSetDefault
        .ContextualAlternates = 0
    End With
    Selection.MoveRight Unit:=wdCharacter, Count:=1
    Selection.MoveRight Unit:=wdWord, Count:=1, Extend:=wdExtend
    Selection.InsertSymbol Font:="Symbol", CharacterNumber:=3332, Unicode:= _
        True
    Selection.MoveRight Unit:=wdWord, Count:=1, Extend:=wdExtend
    Selection.Font.Size = 9
    Selection.MoveRight Unit:=wdCharacter, Count:=1
    Selection.Font.Size = 12
    Selection.TypeText Text:=""
End Sub
```

Shortcut for Recording a New Macro

**Tip**
You can record a new macro by clicking on the **Macro** button on the **Status Bar** at the bottom left of the screen.

![No macros are currently recording. Click to begin recording a new macro.](image)
Macro to number headings in a document in which heading levels have been assigned

It might be quite useful to have a macro to insert heading numbering in a document for two reasons: first because Word has a habit of not remembering to number headings when a document has been saved and closed and then opened again, and secondly if you like to have customised heading numbering rather than one of the built-in default styles.

You need to have already assigned heading levels 1,2,3 etc. to the text (paragraphs) that make up your headings. Also it works better if you are actually in a Level 1 heading when you do this.

To assign number headings with a macro:

- Place your cursor in a Level 1 heading.
- Click on the Developer tab, in the Code group, click on Record Macros, and the Macros dialogue box appears.
- Give the macro a name; e.g. heading_numbering (remember no spaces are allowed in a macro name).
- Click on the OK button. You are now recording.
- Click on the Home tab, in the Paragraph group, click on Multilevel list. If you want one of the standard/default number formats, click on one of the formats under List Library – e.g. select:

```
1 Heading 1
1.1 Heading 2
1.1.1 Heading
```

- Stop the recorder.

Visual Basic code created by Recorder for default heading numbering

If you choose the default heading numbering as used in this document you get miles of code most of which can be edited out. You can delete all the instructions which say ‘undefined’ and the instructions to do with heading levels below three or four, depending on how many heading levels you have in your document. I would suggest that four heading levels should be as many as you need.

Macro to insert the Filename in Footer

We will use the shortcut button to record a macro to automatically put the filename of a document into the footer area, together with the page numbering.
To create a macro to insert the filename in a footer:

- Click on the Macro button on the Status bar at the bottom left of the screen.

- The Macros dialogue box appears type a macro name; e.g. **Custom_Footer**, (don’t worry at this stage about a shortcut key)

- Click on the OK button, everything you do is being recorded.

- Click on Insert tab, in the Header & Footer group, click on Footer and select Edit Footer. The cursor should now be on the left-hand side in the Footer.

- Click on the Insert tab, in the Text group, click on Quick Parts and select **Field...** the Field dialogue box appears:

- Under Categories: click on the arrowhead and select **Document Information**

- Under Field names: select FileName, you can also select a format if you wish and **Add the path to the filename**.

- Click on the OK button to go back to the footer area.

- If you want a border above the entire footer, click on the Home tab, in the Paragraph group, click on the arrowhead to the right of the **Borders** button and select **Top Border**

- Close the **Header** and **Footer** area.

- **Stop** the recorder.
Running Word Macros

Having either recorded or written a macro, you can run it by one of the following methods: Assign the macro to a shortcut key or Developer tab where you view a list of macros, from which you can run.

Using the Developer tab

To run the macro to insert a filename in the footer:

- Place the cursor in the footer and then run the macro. *(click on the Insert tab, click on Footer and select Edit Footer)*
- Click on the Developer tab, in the Code group, click on the Macros button alternatively, press Alt F8, *(the Macros dialogue box appears):*

![Macros Dialogue Box]

- Under Macro name: any macros you have created will be listed, scroll down the list until you come to the Macro you want to run, select it.
- Click on the Run button

Assigning a Macro to a shortcut key

**Danger!**
Be aware that there are hundreds of shortcuts keys already assigned to Windows and Microsoft Office applications. **DO NOT OVERWRITE AN EXISTING SHORTCUT KEY!**

To assign a shortcut key to a macro in Word:

- Click on the File tab *(top left of the screen)*
- Click on the **Options** button to display the **Word Options** window.

- Click on **Customize Ribbon** on the left and under **Choose commands from**: click on the arrowhead and select **Macros**, a list of existing macros are displayed.

- Select the macro that you want to assign to a shortcut key.

- Click on the **Customize** button at the bottom of the window, (*the Customize Keyboard dialogue box appears*).

- Type a shortcut key combination in the **Press new shortcut key**: text box.

- Click on the **Assign** button and then **Close**.

**Danger!**

Do not overwrite an existing shortcut key combination and **BEWARE OF COMBINATIONS SUCH AS ALT+F** because Word will tell you this is **UNASSIGNED** but in fact it is the Windows shortcut key for the **Office** menu.

Also be aware that other applications you may have open while using Word, such as Hypersnap, also respond to keyboard shortcuts and you may not want to clash with these.

### Edit a macro in Word

**To edit a macro in Word:**

- Click on the **View** tab, in the **Macros** group, click on **View macros**, (*the Macros dialogue box opens*) and select the macro you want to edit.

- Click on the **Edit** button to open the VBE – (*for more information and different ways of accessing VBE, see Open the Visual Basic Editor (VBE) on page 18*).

- Click on the **+** next to **Normal**. Click on the **+** next to the **Modules** folder. Select **NewMacros** by **Double-clicking** it. This will open the **Code** window for the Word Macros.
• To close the VBE screen and go back to the Word document, click on the Close button at the top right of the screen, alternatively, press Alt F11.

Storing Macros template in Word

Macros in Word are stored in templates. The default is for macros to be stored in the default template Normal.dotm. If they are there, they are then available to any document in Word. They can also be stored in specific templates; they are then only available in documents using that template. Macros can be copied and moved between templates using Organiser.

Copying macros from one template to another

To copy a macro from one template to another using the organizer:

• Click on the Developer tab, in the Templates group, click on Document Template to display the Templates and Add-ins dialogue box.

• At the bottom left of the Templates and Add-ins dialogue box, click on the Organizer… button. The Organizer dialogue box appears:

• Click on the Macro Project Items tab, you will need to specify the document template you want to copy the macro to. If the template you want is not showing in the Organizer dialogue box, Click on the Close File button. The Close File button will then be replaced by an Open File button, which automatically opens the template folder C:\Documents and Settings\your username\Applications Data\Microsoft\Templates and you can select the template you want.
Select the source template on one side of the screen and destination template on the other side.

Click on the Copy button. This will put a copy of the macro onto the destination template.

Click on the Close button.

Examples of macros in Excel

Macro to put a bullet in a cell in a worksheet

This macro allows you to enter a bullet (●) into any cell in any worksheet quickly and easily.

To create a macro to insert a bullet in a cell:

• Select the cell into which you want to enter a bullet.

• Click on the Developer tab, in the Code group, click on Record Macro, alternatively, click on the Macro button on the Status bar at the bottom left of the screen.

• The Macros dialogue box appears:

• In the Macro name: text box type the macro name; e.g. bullet - no spaces are allowed in the macro name.

• Don’t worry at this point about a shortcut key – unless you want to. Click OK. The recorder is now running and everything you do will be recorded.

• Press and hold down the Alt key type 0149 on the numeric keypad (Num Lock must be on). A bullet should appear both in the cell and in the Formula bar.
Click on the Enter button (the tick) to the left of the Formula bar or press Enter. The bullet is inserted in the cell.

To stop the Recorder, click on the Developer tab, in the Code group, click on Stop Recording, alternatively click on the Macro button at the bottom left of the screen.

Relative or absolute references?
An absolute reference always refers to the same location, while a relative reference works relative to where the cursor is currently positioned. Suppose we place the cursor in the cell A4 and record a macro to move to D4. If the macro is recorded with absolute references then we will always return to D4 when we run the macro. If we record the macro with relative references the cursor will move four cells to the right from the starting cell.

Form controls

Excel allows us to add form controls to a worksheet. By form controls, we mean elements such as tick boxes, buttons to click on, spinners, scroll bars, and lists to pick from. We can have elements on the worksheet that we are more used to seeing as part of a dialogue box. To create the form controls we need to have the forms toolbar displayed on screen.

Display the Developer tab, if it’s not on the Ribbon

- See page 3

- Once the Developer tab is on the Ribbon we can choose which controls we want to add to the worksheet by clicking on Insert in the Controls group
We can also connect the control to a cell on the spreadsheet so that we can use the control to enter a range of data.

**To create a form control**

- Click on the Developer tab, in the Controls group, click on Insert.
- Click on the control required.
- Click and drag on the worksheet to create the control.
- **Right-click** on the control and select **Format Control**... the Format Control dialogue box appears:
  - Click on the Control tab.
  - Set any properties required. This screenshot shows the options for a spin button control.

---

**My option buttons don’t work properly**

To create option buttons successfully, we first need to create an option group. Then we create the option buttons within the option group. If we just create the buttons on the worksheet without an option group (or outside its border) then Excel doesn’t know which set of options the button is supposed to relate to.

---

**Macro to go to the beginning of the next line when entering data in a flat-file database**

When entering data in what is effectively a flat-file database in Excel it is something of a nuisance to have to make several key presses to get to the beginning of the next line each time you wish to enter a new record. You can automate this and make the entire move with one click of the mouse or a keyboard shortcut combination by recording the moves in a macro.
It is probably best to change the way in which Excel behaves when editing and entering data and make the use of the Enter key behave in the same way as clicking on the tick in the formula bar; that is, the cursor remains in the cell which you were editing.

**To create a macro to go to the beginning of the next line when entering data in a flat-file database:**

- Click on the **File** tab, click on the **Options** button to display the **Excel Options** window, click on **Advanced** and under **Editing options** deselect **After pressing Enter, move selection**

- Make sure you are in the last cell of one of your data records having entered the data in the cell.

- Click on the **Macro** button at the bottom left of the **Status** bar

- Give macro a name; e.g. **down_left**, click **OK**. You are now recording.

- Click on **Relative Reference** button.

- Press the **down arrow**, then left arrow as many times as you need to get to the first cell of the next record

- **Stop** the recorder by clicking on **Stop Recording** button at the bottom left of the **Status bar**.

**Visual Basic Code as created by the Recorder:**

```
Sub Down_Left()
'  Down_Left Macro
  ' Macro to go to the beginning of the
  ' next line when entering data in a flat-file database
  ActiveCell.Offset(1, -7).Range("A1").Select
End Sub
```
The Visual Basic Editor

The Visual Basic Editor (VBE) is the window used to produce and edit the code. Arguably it is the most sophisticated programming editor currently available.

The VBE window should show as three parts, the **Project Explorer**, the **Properties Window**, and the **Code Window**.

Open the Visual Basic Editor (VBE)

To inspect the code that the Recorder has produced from your actions do one of the following methods:

- Click on the **Developer** tab, in the **Code** group, click on the **Macros** button, *(the Macros dialogue box appears).*
- Select the **Macro** you want to edit and click on the **Edit** button. Or
- Click on the **Developer** tab, in the **Code** group, click on the **Visual Basic** button
- To display the codes in the **Visual Basic Editor** window, you will need to expand the **Modules** (click the + sign to the left of the **Modules** folder) in the **Project – VBAProject** window and select the **Module** that contains the macro codes – e.g. **Module 1**

![VBAProject (PERSONAL.XLSB)](image)

Or

- Press **Alt F11**.

If the two sections are not visible, follow the instructions for the individual sections.
Code Window

The code window is obviously the most important area. The list box at the top right-hand side of the window allows selection of individual Subroutines and the code produced by the Macro Recorder is displayed in the window.

By default, individual parts of the macro are colour coded, and should look something like the script above.

First comes the start of the Subroutine. Two comment lines in green beginning with a single quote mark follow this. These give the name of the macro and any explanatory information you may wish to add. Next, come the actions, then the final line ending the Subroutine. Note that some lines are inset automatically for ease of reading.

Project Explorer

If the Project Explorer (Project – VBAProject pane) is not visible:

- Click on the Project Explorer button on the VBA Standard toolbar, alternatively, press Ctrl R to display the pane together with the tree structure for the VBA Project associated with the workbook.
- You can expand the Microsoft Excel Objects Folder (++) displays the individual worksheets in the workbook. Expanding the Modules folder displays the icon for the code Module1. Double clicking on the module will display the Code Window for the module.
• Note that the workbook appears as a VBA Project. This is a new term intended to bring the worlds of Visual Basic and Excel together! The Project (i.e. workbook) also has any attached VBA modules inside it and is associated with any loaded add-ins, such as the XY Chart Labeler.

• Any workbooks in the XLSTART folder and your personal StartUp folder (if you have one) will also appear in the Project Explorer list (even if the workbooks are hidden). In particular this means that you will see the special Personal.xlsb workbook which Excel creates as a general-purpose storage area for macros, and which is stored in the XLSTART folder.

Properties Window

Immediately below the Project Explorer is the Properties Window that displays properties of any selected item in the Project Explorer.

If the Properties window is not visible:

• Click the Properties Window button on the VBA toolbar

• This window displays the properties of any object selected in the Project Explorer window. Editing in the “Name” box allows you to change the name of the project.
Customising the VBA Environment

When VBE is active you can customise the environment:

- Click on the **Tools** menu and select **Options** to display the **Options** dialogue box which consists of four tabs; **Editor**, **Editor Format**, **General** and **Docking**.

- Under the **Editor** tab it is best to enable the **Auto Syntax Check** option which will display a dialogue if any syntax errors occur while entering code.

- **Auto List Members** displays a list that contains information that logically completes the statement you are typing.

- The **Editor Format** tab allows settings for the display of code including font and text colour.
Object Browser

This tool allows you to look through all the available VBA Objects.

To display the Object Browser:

- When the VBE is active, click on the View menu, select Object Browser, alternatively, press F2, or click on the Object Browser button on the toolbar.

- The top drop-down list contains all the available object libraries from which you can select Excel or the appropriate Office libraries.

- The second drop-down is a search. Enter the search string (e.g. format) and click the binoculars icon. The Search Results window displays all items in the object library that contain “format”.

- Selecting an item and pressing F1 or clicking on the gives context sensitive help.
Creating a Macro by typing it into the Visual Basic Editor

If you have typed the title, subject, or name of your workbook in cell A1 of the worksheet, you can ask Excel to pick up the text that is stored in A1 and make it the filename for the workbook. This is not a macro you can record; you need to type it in from scratch in the Visual Basic Editor.

Open the Visual Basic Editor (VBE)

To open the VBE - Choose one of the following methods:

- Click on the Developer tab, in the Code group, click on Visual Basic

Or

- Press Alt F11 on the keyboard, to open the VBE screen.

To type a macro directly into VBE:

- Open VBE by following one of the options above.
- Under the Project – VBA Project pane, select the book where you want to put the Macro.

- Click on the Insert menu and select Modules to insert a new Module sheet.

Type the following macro code to make Excel pick up the text in a cell and make it the filename.

```vba
Sub SaveFile()
    Dim ThisFileName As String
    ThisFileName = Range(“A1”).Value
    ActiveWorkbook.SaveAs Filename:=ThisFileName
End Sub
```

The finished code should look like this when displayed in VBE.
- Click on the Close button to close the VBE screen, and then run the macro. Excel will pick up the text that is stored in cell A1 and make it the filename for the workbook.

**Edit a macro or User-defined Function in Excel**

To edit a macro or User-defined function in Excel choose one of the following methods:

- Open the VBE – see Open the Visual Basic Editor (VBE) on page 16 - e.g. quick shortcut, press Alt F11.
- Click on the next to the workbook that contains the macros. Click on the next to the Modules folder. Double-click the module to edit.

Or

- Click on the View tab, in the Macros group, click on Macros and then click on View Macros to display the Macros dialogue box, select the Macro you want to edit and click on the Edit Button.
Running macros in Excel

Running a macro in Excel is very similar to the way it is done in Word (see page 9). Having either recorded or written a macro, you can run it by using one of the following options:

- Click on the **Developer** tab in the **Code** group, then click on **Macros** and select your macro from the list and click on the **Run** button
- Assign your macro onto a button on the Quick Access Toolbar and click on the button.
- Assign your macro onto a button on the worksheet and click on it
- Assign a shortcut key combination to the macro.

Adding macros to the Quick Access toolbar

This is very similar to the way it is done in Word.

**To assign a macro to a button and display it on the Quick Access toolbar:**

- Click on the **File** tab *(at the top left of the screen)*
- Click on the **Options** button, the **Excel Options** window appears.

- Click on the **Quick Access Toolbar** button in the left pane
Under **Choose Commands from:**

- Click on the **arrowhead** and select **Macros** a list of macros is displayed, select the macro you want to add to the **Quick Access Toolbar** and click on the **Add** button.

- The macro default symbol is ![default symbol] you can change this symbol if you wish by selecting the macro.

- Click on the **Modify...** button, *(the Modify Button window appears)* with lots of different symbols.

- Select a **symbol** then, click on the **Ok** button
Attaching a macro to a button on a worksheet

In Excel you can also run a macro by attaching it to a button directly on the worksheet. This can sometimes be very convenient and the button remains on the worksheet when you close the file and open it again.

To attach a macro to a button on a worksheet:

- Click on the Developer tab, in the Controls group, click on Insert and draw a button on the worksheet.
- The Assign Macro dialogue box opens automatically.
- Select the appropriate macro name and click OK.

The button remains selected allowing you to change the label from the default Button1 to something appropriate.

- Click off the button to deselect it and then test the macro, by clicking on it.
- To rename the button right-click on the button and select Edit Text.
Assigning a shortcut key to a macro in Excel

To assign an existing macro to a shortcut key:

- Click on the Developer tab, in the Code group, click on Macros, select the macro and click on the Options button, the Macro Options dialogue box opens. You can type in a shortcut key combination.

Danger!
Please see page 9, Assigning a macro to a shortcut key.
Be aware as Excel does not warn you if you are overwriting an existing keyboard shortcut of any kind; e.g. in Excel, in Windows, in another application, such as Hypersnap.

Creating a macro to resize a graph

We are going to create a macro that will resize a graph, and modify the code manually so that the plot area of the graph is square.

To record the macro:

- Select a graph, creating it first if necessary.
- Click on the Developer tab, in the Code group, click on Record Macro. The Record Macro dialogue box appears. Name the macro Resize_Graph.
- Select the Plot Area.
- Resize the Plot Area horizontally and vertically from one of the corner handles.
- Stop the recorder.

To modify the macro

- Start the visual basic editor with ALT-F11.
- Use the Project Explorer to navigate to the module with the macro code.
- Add the following three lines above the ActiveChart.PlotArea.Select line:
  
  ActiveChart.Parent.Width = 350
  ActiveChart.Parent.Height = 350

- Change the numbers on the ‘Selection.Width =’ and ‘Selection.Height =’ lines to be 250.

The final thing we need to do is some error trapping, to get around the problem of what would happen if the macro was run and no chart was selected.
To trap errors:

- Below the last line of comments add the line:
  
  `On Error GoTo ErrorHandler`

- Above the End Sub statement add the two lines:
  
  `Exit Sub
ErrorHandler: MsgBox (“Select a graph first”)`

- Save the code.
- The finished code should look like this:

  ```vba
  Sub Resize_graph()
     ' Macro to resize graph
     On Error GoTo ErrorHandler
     ActiveChart.Parent.Width = 350
     ActiveChart.Parent.Height = 350
     ActiveChart.PlotArea.Select
     Selection.Width = 250
     Selection.Top = 35.077
     Selection.Height = 250
     Exit Sub
     ErrorHandler: MsgBox (“Select a graph first”)
  End Sub
  ```

Where should I store the macro?
We have the choice of storing the macro in this workbook, a new workbook or in a personal macro workbook. If we store the macro in a personal macro workbook the macro will be available whenever we have an Excel file open. If we store the macro in this workbook or a new workbook then it will only be available when that particular workbook is open.

Storing Macros in Excel

Macros in Excel are stored in workbooks and are available to every worksheet within a workbook. All workbooks that contain macros must be saved with macros enabled, which is a file with an XLSM extension. Excel file format default is XLSX extension, this file cannot contain macros. If you try to same this file with macros, Excel will display the following warning dialogue box.
Where to Store the Modules

When a macro is recorded or written the default is to save it in the current workbook from where it can be used in that workbook. For a VBA module to be available to all Excel workbooks it must be in a workbook that is currently open. This can be achieved by storing the modules in one of three locations.

- The Personal.xlsb workbook.
- A workbook located in the XLStart folder.
- An Add-In loaded using the Add-In Manager.

Any open workbooks, (both hidden and visible) appear in the Project Explorer where their properties and code can be edited.

The Personal.xlsb Workbook

Personal.xlsb is a special hidden workbook created the first time a macro is recorded. This special workbook, together with any other book(s) in XLSTART or your alternative start-up folder, open automatically when Excel is loaded. Personal.xlsb is convenient but as it is commonly a target for virus writers (notably the Laroux variants) it is not particularly safe.

When you record a macro you are given the option to store it in the “Personal Macro Workbook” i.e. Personal.xlsb.

Note
If you save your macros in the Personal Macro Workbook, which is hidden by default. You must unhide it by using the Unhide command in the View tab, before you can edit the macro.

Deleting Macros in Word and Excel

Deleting a Macro in Word

To delete a macro in Word:

- Click on the Developer tab, in the Code group, click on Macros, alternatively, press Alt F8 to display the Macros dialogue box.
• Select the **macro** you want to delete and click on the **Delete** button, the following dialogue box appears, click on the **Yes** button to delete the macro.

![Dialogue box for deleting a macro](image)

**Deleting a Macro in Excel**

• To delete a macro in Excel it's the same as deleting a macro in Word, follow the steps above.

**User-defined Functions in Excel**

Using Visual Basic for Applications (VBA) it is possible to write user-defined functions that behave in the same way as standard Excel functions. They are not macros but you type them in the Visual Basic Editor in the same way as you would type in a macro from scratch without using the Recorder.

The basic structure is:

```vba
Function name ([arglist])[As type]
  [statements]
  [name = expression]
  [statements]
  [name = expression]
End Function
```

Items in square brackets […] are optional. Words in Bold are required.

<table>
<thead>
<tr>
<th>Name</th>
<th>The name given to the function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arglist</td>
<td>List of arguments passed to the function (arguments are separated by commas).</td>
</tr>
<tr>
<td>Type</td>
<td>Data type of the value returned by the function.</td>
</tr>
<tr>
<td>Statements</td>
<td>A valid Visual Basic statement.</td>
</tr>
<tr>
<td>Expression</td>
<td>Expression to set the value returned by the function.</td>
</tr>
</tbody>
</table>

We will create some simple user-defined functions: one to calculate the area of a triangle, given the length of two adjacent sides and the included angle and two others to convert degrees Celsius to Fahrenheit and vice versa.
User-defined function to calculate the area of a triangle

As an example we will produce a user-defined function to calculate the area of a triangle given the length of two adjacent sides and the included angle:

To create a user-defined function to calculate the area of a triangle:

- Set up a worksheet with the data below:

  \[ \text{Area} = \frac{1}{2} ab \sin(\theta) \]

  ![Worksheet with data](image)

- D4 contains the formula \(0.5 \times A4 \times B4 \times \sin(\text{RADIANS}(C4))\) to test the user defined function.

- Press Alt F11 to open the Visual Basic Editor and click on the Insert menu and select Module.

- Type in the code below. As you type the keywords the Auto List Members option will supply a list of appropriate options to complete the statement.

- Type in the comment line starting with an apostrophe.

  `'Calculates the area of a triangle given two sides and the included angle'`.

```vba
Function TRIAREA(sideA, sideB, Angle)
    Alpha = Application.WorksheetFunction.Radians(Angle)
    'Deg to Radian
    TRIAREA = 0.5 * sideA * sideB * Sin(Alpha) 'Calculate area
End Function
```
The comment line, shown in green, is preceded by an apostrophe.

The code starts with keyword **Function** (shown in blue) followed by the name of the user defined function and its three arguments. These will be shown in the **Insert Function** dialogue box as typed. The calling formula passes the values of its arguments to the function by position rather than by name.

- The following line is an assignment statement where the variable **Alpha** is given a value. As the angles are entered in degrees in the worksheet they must be converted to radians before the value is used in a trigonometric function. Here the worksheet function **Radians** has been used, hence the keywords **Application.WorksheetFunction**.

- In the final assignment statement, the function is given a value that is returned to the worksheet. The trigonometric function sine is available in Visual Basic and does not require the **Application.WorksheetFunction** qualifier.

- The final line contains the obligatory **End Function** (shown in blue).

- Save the codes and return to the Excel worksheet and enter the formula using the newly created function in **E4**.

**To insert the function into cell E4:**

- Click on the **Insert Function** button on the **Formula bar**, the **Insert Function** dialogue box appears:
You should find TRIAREA in the Insert Function dialogue box under User-Defined functions.

Scroll down the function list and select the function – e.g. TRIAREA.

Click on the OK button to display the Function Arguments dialogue box.

Enter the values from the worksheet in each text box and click on the OK button.
The **Formula** and **Function** columns should agree.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Function TRIAREA to compute the area of a triangle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Side A</td>
<td>Side B</td>
<td>Angle</td>
<td>Formula</td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>90</td>
<td>1.000000</td>
<td>1.000000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>60</td>
<td>1.732051</td>
<td>1.732051</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>72</td>
<td>1.902113</td>
<td>1.902113</td>
<td></td>
</tr>
</tbody>
</table>

**Add a Description to Function Arguments Dialogue box**

- Although the **Function Arguments** dialogue box shows the arguments of the function, the description is incorrect (it tells you there is no help available).

**To add a description:**

- Activate the worksheet in the workbook containing the **User-defined** function.
- Press **Alt F8** to display the **Macro** dialogue box.

- In the **Macro Name**: text box, type the function name – e.g. **TRIAREA**.
- Click on the **Options...** button and type a description in the **Macro Options** dialogue box.
• Click on the OK button to close the Macro Options dialogue box.
• Click on the Cancel to close the Macros dialogue box.
• Check the Insert Function dialogue box and notice the description has changed.

User-defined functions to convert degrees Fahrenheit to Celsius and vice versa

You can enter the following user-defined functions into the VBE:

• Open the VBE as before, e.g. Alt F11, insert a module sheet from the Insert menu and in the code window on the right-hand side, type the following:

  Function Cels(Fahren)
  Cels = (Fahren - 32) * 5 / 9
  End Function

• Now below Cels(Fahren) enter the code for the second function:

  Function Fahren(Cels)
  Fahren = (Cels * 9 / 5) + 32
  End Function
Functions available in VBA

There are three types of function available in VBA:

- Worksheet functions provided by Excel
- Built-in functions provided by VBA
- Custom functions (User-defined)

You can use the Office Assistant to find the worksheet functions available to Visual Basic by clicking the Help button and searching for "worksheet functions". Not all worksheet functions are available as many are provided as VBA built in functions. These run more efficiently than worksheet functions.

Perversely the VBA functions are more difficult to find. Activate the Object Browser (see page 19) and search, for example, on format. A list of VBA formatting functions appears.

Currently the user-defined function is available in any sheet of the current workbook.

Using Functions in Other Workbooks

User-defined functions are available in the workbook where they were coded and in every worksheet in that workbook.

There are a number of ways to access functions in other workbooks:

- Paste the code into another worksheet.
- If a workbook containing a user-defined function is open that function is available to other workbooks. Any workbook placed in the XLStart folder (C:\Program Files\Microsoft Office\Office\XLStart) is opened automatically when Excel is opened. Consequently if the workbook containing user-defined function(s) is put into XLStart the problem is solved.
- The module sheet containing the function can be converted into an Add-In.

Note
If you wish to access the function from another workbook the name of the workbook containing the function must be included.

e.g. = MyFuncs.xls!TRIAREA(A4,B4,C4)

This is not necessary if the function is installed as an add-in.
A User-Defined Workbook

The most convenient storage method is to create an auto-loading workbook that is also hidden. Hidden workbooks can be edited from the Visual Basic Editor.

How to create a hidden user-defined workbook:

- Open the workbook containing your macro(s).
- Save it as, for example, MyMacros in C:\Program Files\Microsoft Office\Office11\XlStart.
- Click on the View tab, in the Window group, click on Hide.
- Exit Excel and save any changes to MyMacros if prompted.

Macro Virus Protection

There are now more than 3000 macro viruses written using VBA. Many commercial virus-checking programs will search for and disinfect Macro viruses; however, the first line of defence is only to accept workbooks and documents containing macros from a trusted source. If you attempt to open an Excel workbook that contains a macro, you will get the warning shown below:

![Security Warning](image)

The virus payload can be a simple message, a replication of itself (as in the most “successful” Excel virus, Laroux), cause corruption of your files, or re-format your computer’s hard drive. The Word virus “Switcher”, for example, attempts to delete *.xls (Excel) and *.mdb (Access) files (among others) and the Excel Melissa clone Papa virus posts itself to the first 50 contacts in your Outlook addresses book.
To adjust the security level for files that might contain macro viruses

- Click on the Developer tab, in the Code group click on Macro Security.
- Select the required option.

References

As an excellent starter try the unfortunately named:

Excel VBA Programming for Dummies, by John Walkenbach, published by Hungry Minds Inc.

More advanced references are:


Excel 2007 Power Programming with VBA (Mr. Spreadsheet’s Bookself), by John kWalkenbach.