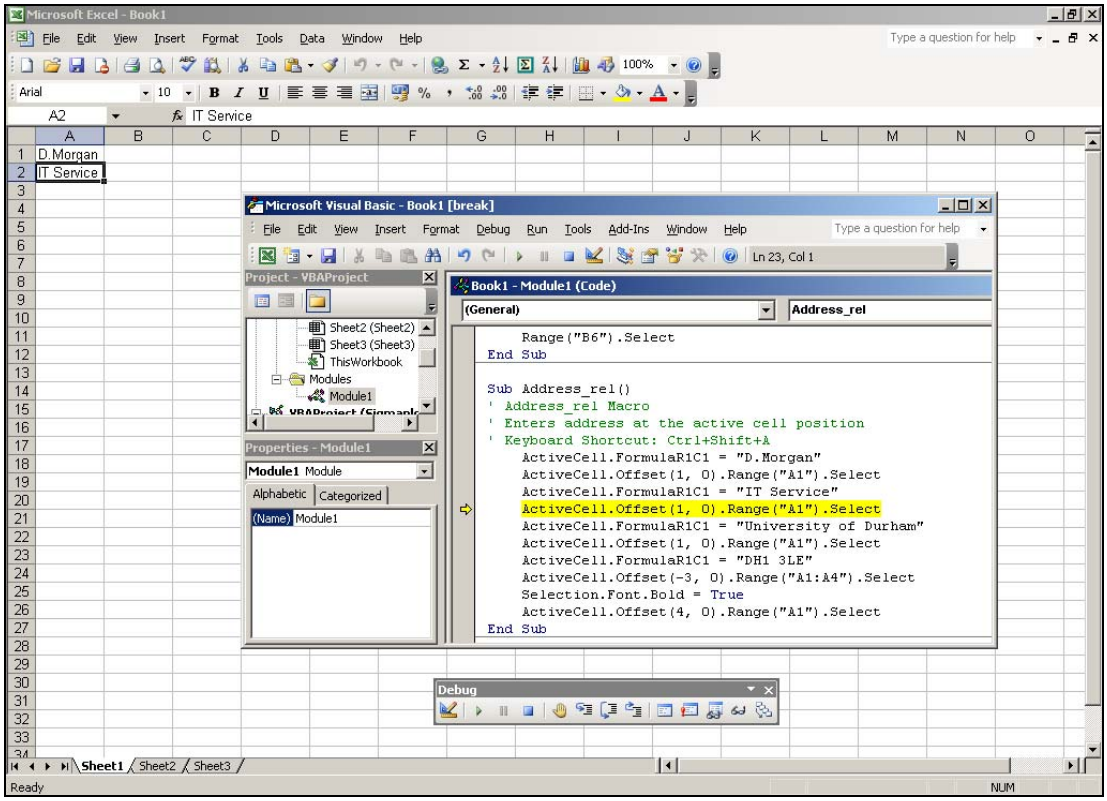
ITS Information Technology Service

Guide 39

Version 1.0

Introduction to using macros in Microsoft Excel 2003

This guide provides an elementary introduction to using macros in Excel 2003. Instructions are given for recording macros, and for writing simple macros in Visual Basic for Applications.



Durham University

£1.50

Document code: **Guide 39**
Title: **Introduction to using macros in Microsoft Excel 2003**
Version: **1.0**
Date: **June 2006**
Produced by: **University of Durham Information Technology Service**

Conventions:

In this document, the following conventions are used:

- A typewriter font is used for what you see on the screen.
- A **bold typewriter font** is used to represent the actual characters you type at the keyboard.
- A ***slanted typewriter font*** is used for items such as filenames which you should replace with particular instances.
- A **bold font** is used to indicate named keys on the keyboard, for example, **Esc** and **Enter**, represent the keys marked Esc and Enter, respectively.
- A **bold font** is also used where a technical term or command name is used in the text.
- Where two keys are separated by a forward slash (as in **Ctrl/B**, for example), press and hold down the first key (**Ctrl**), tap the second (**B**), and then release the first key.

Contents

1 Introduction

In Excel, you can automate tasks by using macros. A macro is a set of instructions that tells Excel what to do. These commands are written in a computer programming language called Visual Basic for Applications (VBA).

This document first explains how to create macros using the Macro Recorder provided by Excel. Just as a tape recorder can be used to record sounds which can be played back later, so the Macro Recorder can record your actions into a macro. In this way you can create macros without learning about Visual Basic. These macros can be run whenever you wish, automatically repeating your recorded actions and so saving you time and effort.

Later in the Guide you will learn a little about Visual Basic and try writing macros from scratch by entering instructions into a module. Solutions to these exercises are given in section 16.2.

Macros are also referred to as *procedures* — the two terms mean the same thing.

2 Recording and running macros

When using the Macro Recorder it is a good idea to practise the steps you want to take *before* you start recording. In this way you can avoid recording mistakes, and the subsequent corrections, into the macro. Excel does try to be helpful and will not record an action until you complete it. So, for example, a cell is not recorded as being selected until you perform some action in that cell. Also, the recorder does not record a menu command that brings up a dialog box unless and until you press OK in that dialog box.

Each time you record a macro, the macro is stored in a module attached to a workbook. Recorded macros can be stored in **This Workbook** (the current workbook), **New Workbook** (a new workbook) or the **Personal Macro Workbook**. Code recorded in **personal.xls,** the Personal Macro Workbook, is available when you start Excel. Macros recorded in other workbooks are available whenever their workbooks are open (this includes using them from another workbook).

In readiness for recording a macro:

1. Activate Excel.
2. Use the new workbook that is opened for you.

2.1 Recording a macro which uses absolute cell references

Try recording a macro to produce your name and address as follows:

1. On the **Tools** menu, point to **Macro**, and then select **Record New Macro**.
2. In the **Macro name:** box, enter **Address_abs** as the name of the macro.

The first character of a macro name must be a letter. Other characters can be letters, numbers, or underscore characters. Spaces are not allowed in a macro name but an underscore character works well as a word separator.

3 Leave the **Shortcut key:** box blank (this will be discussed later).

4 In the **Store macro in:** box, leave the setting at **This Workbook**.

5 Select the text already in the **Description:** box and type

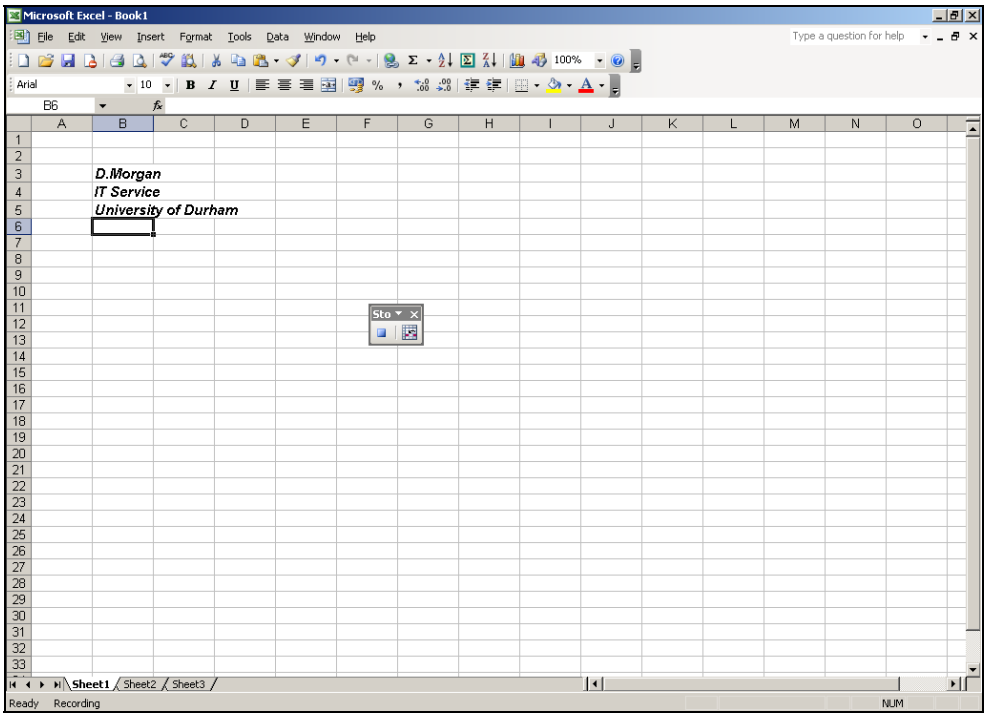
Enters address starting in cell B3

6 Click **OK**.

7 A **Stop Recording** toolbar will appear. Drag it to a new position if it is in the way.

8 On the **Stop Recording** toolbar, make sure that the **Relative Reference** button has *not* been "pressed in". There should *not* be a shaded border around it.

9 On **Sheet1**, click in **B3** and type in your name. Enter your address in the cells below, as shown in the example.



10 Make the text Bold Italic.

11 Click in **B6** (to remove any highlighting of selected cells).

12 On the **Stop Recording** toolbar, click the **Stop Recording** button.

A macro has now been recorded. Cells that were selected during the recording of the macro were given absolute cell references. Consequently, regardless of which cell on your worksheet is active when you run the macro, your name and address will always be created in the same position on your worksheet.

Note: You can choose to use relative cell references when recording a macro. This will be dealt with in section 2.3.

2.2 Running a macro using the Macro dialog box

Now run the macro from Sheet2:

1. Activate **Sheet2** and click on any cell other than **B3**.
2. On the **Tools** menu, point to **Macro**, and then click **Macros**.
3. Click on **Address_abs** in the list of available macros (probably the only one offered at present).
4. Click **Run**.

Your address will be entered on Sheet2 just as it was on Sheet1.

Note: If you want to interrupt a macro before it has finished, press **Esc**.

2.3 Recording a macro which uses relative cell references

The **Address_abs** macro used absolute cell references. The next macro to be recorded will enter your name and address at whichever cell position is active when the macro is run. Since the macro has to select cells relative to the position of the cell that is active when the macro is run, the macro recorder will have to record relative cell references.

1. Activate **Sheet1**.
2. Click in cell **B11**.
3. On the **Tools** menu, point to **Macro**, and then select **Record New Macro**.
4. In the **Macro name:** box, type

   Address_rel

   as the name of the macro.
5. In order to assign a key combination to the macro, type a capital

   A

   in the **Shortcut key**: box which will then show **Ctrl+Shift+A**.
6. In the **Store macro in:** box, leave the setting at **This Workbook**.
7. Replace the text already in the **Description:** box with

   Enters address at the active cell position
8. Click **OK**.
9. A **Stop Recording** toolbar will appear. Drag it to a new position if it is in the way.
10. On the **Stop Recording** toolbar, click the **Relative Reference** button. It will appear to be “pressed in” with a border around it.

Microsoft Excel will continue to record macros with relative references until you quit Microsoft Excel or until you click the Relative Reference button again.

11 Type your name and address in **B11, B12, B13** and **B14**. This time, include the postcode so that this macro produces different text from the first one.

D.Morgan
IT Service
University of Durham
DH1 3LE

12 Make the text Bold.

13 Click in **B15** (to remove any highlighting of selected cells).

14 On the **Stop Recording** toolbar, click the **Stop Recording** button.

Notes:
If you want a macro to select a specific cell, perform an action, and then select another cell relative to the active cell, you can mix the use of absolute and relative references when you record the macro.

To use Relative References while recording a macro, make sure that the Relative Reference button is pressed in.

To record with absolute references, make sure that the Relative Reference button is *not* pressed in.

2.4 Running a macro using a shortcut key

The **Address_rel** macro could be run as described in section 2.2. Since a shortcut key was assigned to it, try using that instead:

1 Activate **Sheet2** and click on a cell (for example, **H14**).

2 Press the **Ctrl**, **Shift** and **A** keys together.

Your name and address should appear in and below that cell.

3 Try running the macro again from somewhere else on **Sheet2**.

3 Making a macro easier to use

There are other ways of running a macro. A macro can be assigned to:

- a drawing object on a worksheet or chart
- a button on a toolbar
- a command on one of Excel's menus

3.1 Running a macro from a drawing object on a worksheet

A macro can be assigned to a drawing object on your worksheet.

1 Click on **Sheet3,** which should be blank.

2 From the **View** menu, select **Toolbars** and then **Drawing** (unless the Drawing toolbar is already displayed).

3 Select the **Oval** button and draw an oval somewhere on your worksheet. Make the width of the shape a little bigger than the width of a typical cell.

4 To put some text on your shape, right-click on the oval you have drawn and select **Add Text.**

5 Type

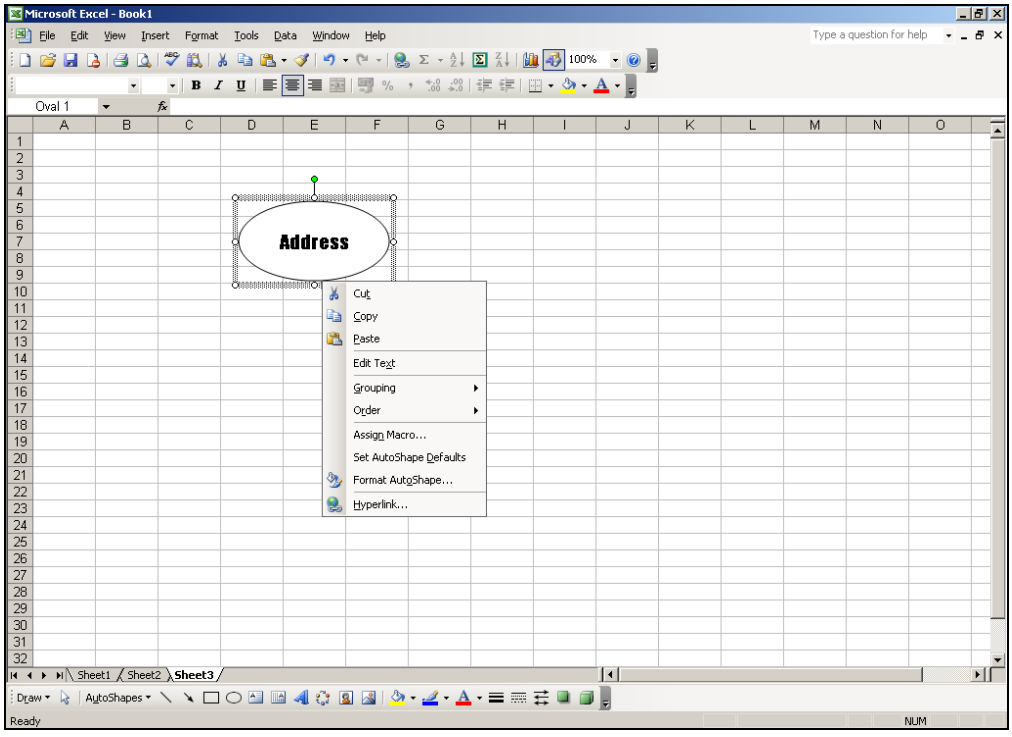
Address

6 Click away from the oval.

7 If the oval is too wide/narrow for the text, select the oval and drag one of its sides to change the shape.

Next, assign a macro to the button:

8 Right-click on the oval (be careful to select the oval and not the text box).



9 From the shortcut menu, select **Assign Macro**.

10 In the **Assign Macro** dialog box, select the macro **Address_rel.**

11 Click **OK**.

Now test the button.

12 Click in any cell (**J13** for example).

13 Click on the **Address** button.

Your address should appear wherever you clicked on your worksheet.

Notes:
If, having created a button, you want to move it to another position on the worksheet, you should use a *right*-mouse click to select it and then drag it (a left-click activates the macro).

If you ever wish to change the macro associated with a particular button, you should right-click on the button, select **Assign Macro** and choose a different one.

3.2 Running a macro from a button on a toolbar

A macro can also be run from a button on any of the built-in toolbars or on a custom toolbar.

As an example, assign your **Address_abs** macro to a Smiley Face button on a custom toolbar as described below.

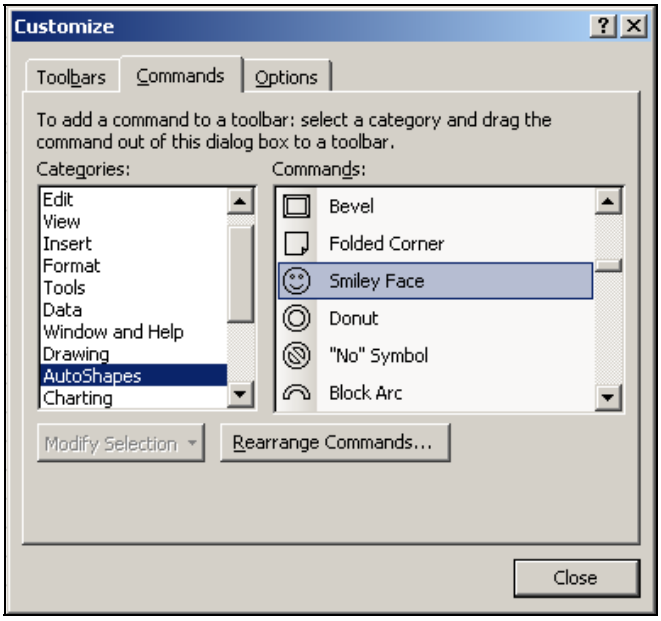
1. Move the mouse so that its pointer is on one of your toolbars.
2. Click the right mouse button.
3. From the shortcut menu, select **Customize**.
4. In the **Customize** dialog box, select the **Toolbars** tab.
5. Click on the **New** button.

A **New Toolbar** dialog box will appear.

6. In the **Toolbar name:** box, type

   Macros

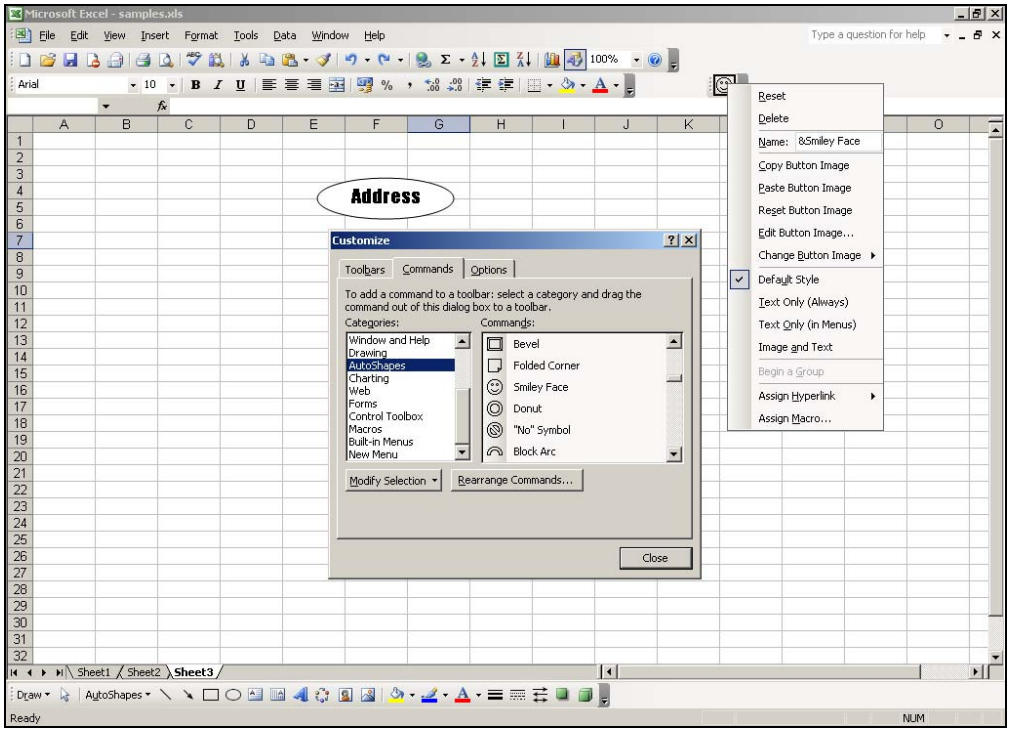
   and click **OK**.
7. Drag the new toolbar (just a small rectangle at this stage) to a clear region near your other toolbars.
8. In the **Customize** dialog box, click on the **Commands** tab.
9. In the **Categories**: box, select **AutoShapes**.
10. In the **Commands**: box, scroll down until you can select the **Smiley Face**.



11 Drag that **Smiley Face** button to your new **Macros** toolbar.

12 Right-click on the **Smiley Face** button.

Note that the shortcut menu includes options to change the button's image and edit it.



13 Click on **Assign Macro**.

14 In the **Assign Macro** dialog box, select the **Address_abs** macro and click **OK**.

15 In the **Customize** dialog box, click on **Close**.

Note: A custom toolbar belongs to the workbook that is active when the toolbar is created.

Now try using the button you created:

1. Clear the cells containing your name and address on **Sheet2**.
2. Click on the **Smiley Face** button on your custom toolbar.

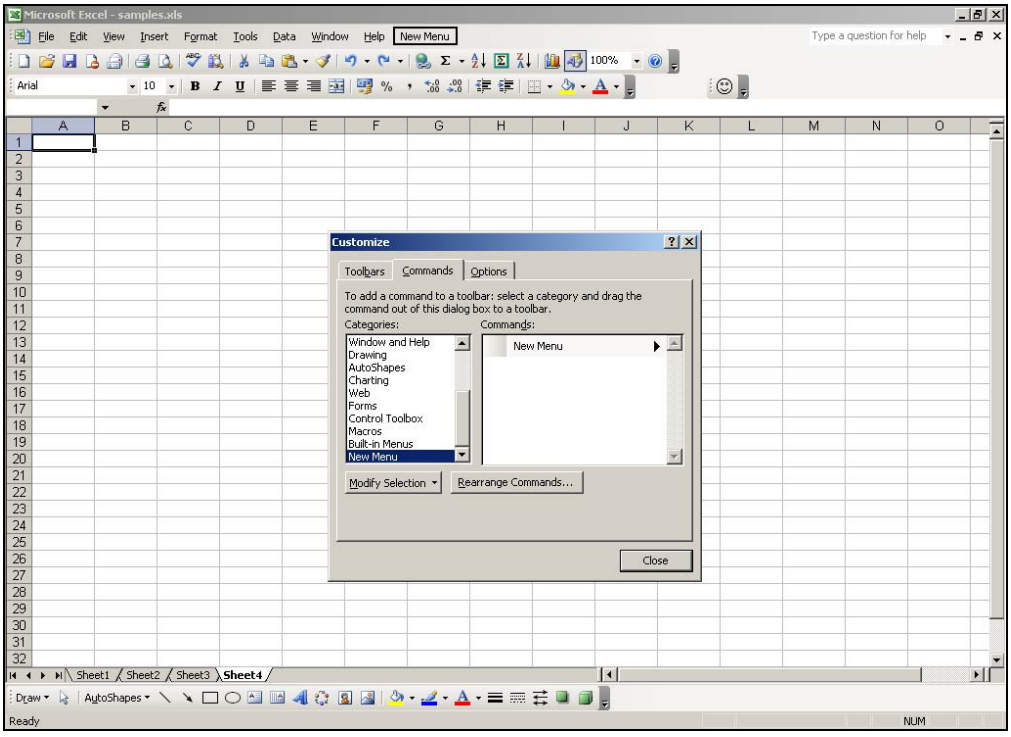
Your name and address should appear, starting in cell **B3**.

3.3 Running a macro from a command on one of Excel's menus

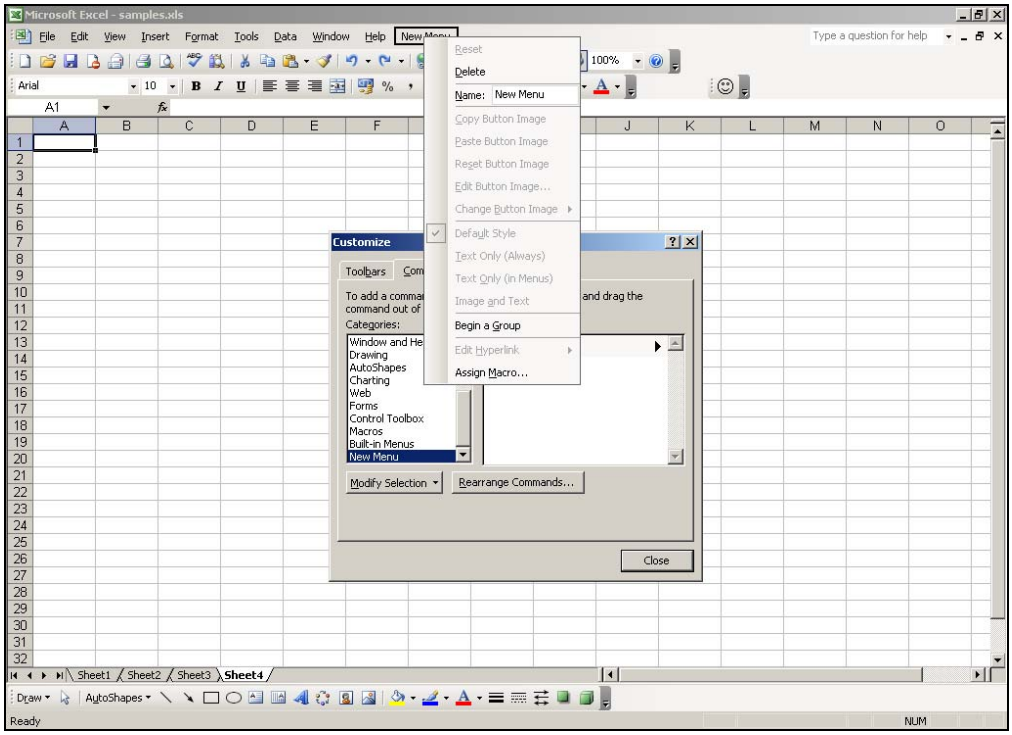
A menu command can be added to a menu so that selecting that new command will run a macro.

As an example, a new command called **Work Address** can be put on a new menu called **Macros** on the standard toolbar as follows:

1. Make sure that the workbook containing your macros is open.
2. Insert a new worksheet using **Insert** | **Worksheet** (probably **Sheet4**).
3. From the **Tools** menu, select **Customize**.
4. In the **Customize** dialog box, select the **Commands** tab.
5. Scroll down the items in the **Categories**: box and select **New Menu**.
6. Point to the **New Menu** item in the **Commands**: box and drag it to the standard toolbar just to the right of **Help**.



7. Right-click on the **New Menu** item on the menu bar.



8 Select the text in the **Name**: box and type

&Macros

9 Press the **Enter** key.

The ampersand (&) in front of the **M** indicates that **M** is the accelerator key for that menu (the underlined letter).

Next, create a new menu command on the **Macros** menu:

1 In the **Categories**: box (in the **Customize** dialog box — still on the screen), select **Macros**.

2 From within the **Commands:** box, drag the **Custom Menu Item** up to the **Macros** menu and *keep the mouse button held down*.

An empty drop-down menu will appear underneath the command.

3 Drag the **Custom Menu Item** into that blank region and let go of the mouse button.

4 Right-click on **Custom Menu Item** in the **Macros** submenu.

5 Change the **Name:** setting to

&Work Address

The position of the **&** specifies that **W** is the accelerator key.

6 Select **Assign Macro** (at the bottom of that box).

7 In the **Assign Macro** dialog box, select the **Address_rel** macro and click **OK**.

8 In the **Customize** dialog box, click **Close**.

A new menu and its commands are stored with the workbook that is active when they are created and will appear automatically whenever that workbook is opened.

Test this new command:

1. On your inserted worksheet (probably **Sheet4**), click in **D4**.
2. Select **Work Address** from the **Macros** menu.

Your address should appear at that active cell position, **D4**.

Now, test the accelerator keys:

1. On **Sheet4**, click in **G4**.
2. Press **Alt**, then **M** and then **W**.

Again, your address should appear at the active cell position, **G4**.

3.4 Changing macro options

If you need to change the options of a particular macro, the first step is to go to the **Tools** menu, select **Macro** and then **Macros**.

Next, you should click on the name of the macro whose options you wish to change and click the **Options** button.

Then you can assign (or change) the **Shortcut key** or change the description of the macro in the **Description** box. Finally, click on **OK**.

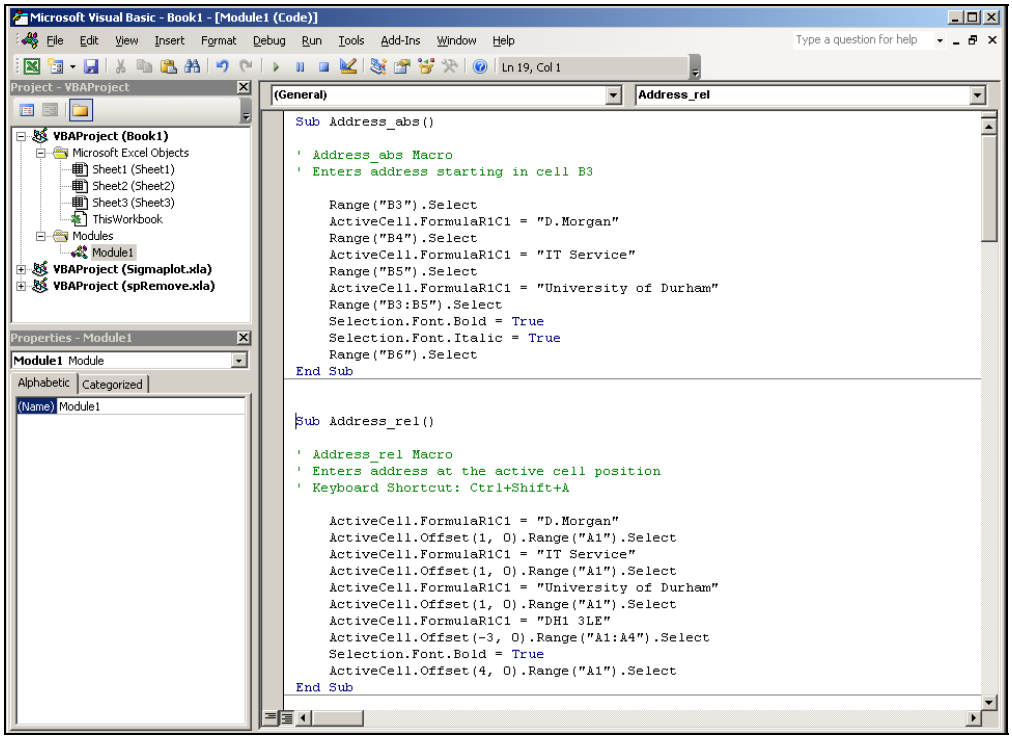
4 Editing a macro

When you recorded your first macro, Excel created a module in the active workbook. This module contains instructions written in Visual Basic for Applications code. In order to view the module:

1. From the **Tools** menu, select **Macro** and then **Macros**.
2. Select **Address_abs** and click the **Edit** button.

An application called the Visual Basic Editor will be activated.

This shows a bewildering amount of information but for the moment just look at the **Book1 - Module1 (Code)** window. When maximised it appears as shown below.



4.1 General form

Special Visual Basic terms, called **keywords**, are displayed in **blue**. Every macro starts with **Sub** (followed by the name you gave the macro and a pair of brackets) and ends with **End Sub.**

The lines in **green** that start with apostrophes are **comments**. These don't affect what the macro does and can be changed. The name of your macro and the description you typed in when recording the macro appear as comments. It is a good idea to add comments throughout a macro describing what is being done at each stage. Then, whenever you look at a macro you can quickly understand it.

The other lines are **black** indicating that they are statements in Visual Basic. Some lines are indented to make it easier to see the structure of the macro.

An underscore (_), known as a **line-continuation character**, sometimes occurs at the end of a line, where it is used to indicate that the code on the next line is still part of the same logical line.

When you record a more complicated macro than this first one, you will probably find that some of the code is not essential. Excel records everything in great detail, and includes all the arguments even when default settings are used.

4.2 Making changes

The **Book1 – [Module1 (Code)]** window is rather like a window for word processing and it is easy to make changes to your macro. You could, for example, change the initial of your first name to the name itself (D.Morgan to Josephine Coleman) in the **Address_abs** macro.

1. Change your name in some way and leave the Microsoft Visual Basic window open.

Now run this new version of the macro as follows:

2. *Either*

   From the **Run** menu, select **Run Sub/UserForm**

   or

   Press the **F5** key.

Next, check this has worked:

3. Return to your worksheet in one of the following ways
   - Click the **Microsoft Excel** button on the Task bar.
   - From the **View** menu, select **Microsoft Excel.**
   - Press the **Alt** and **F11** keys together.
4. Check that your new name and address have appeared, starting in cell **B3**.

The Microsoft Visual Basic window is still available from the Task bar.

If you are in the Visual Basic window and want to *close* that window as you return to Excel, select **Close and Return to Microsoft Excel** from the **File** menu (or press the **Alt** and **Q** keys together).

Recording a macro and then looking at it is a good way of learning about some of the commands. At some stage you will probably want to write your own macro or at least add some lines to an existing macro. Recorded code is not always the ideal code for what you want to do. It usually deals with a specific worksheet or range of cells and doesn't repeat actions (unless you run it more than once). So, you may need to add Visual Basic decision-making and looping structures.

In the sections that follow, you will find more information about Visual Basic for Applications.

5 Visual Basic Grammar

5.1 Objects

Visual Basic is an object-oriented language. This means that all the items in Excel are thought of as objects. There are more than a hundred of them. Examples of objects are:

- the Excel application (the largest object)
- a workbook
- a worksheet
- a range
- a chart
- a legend

You may like to think of an object as a noun (just as **cake** is a noun). In your macro, **Range("B3")** is an object.

An object can contain other objects. The Application object is at the top level. Any changes that you make to the Application object affect the whole application. The Application object contains other large objects such as Workbooks. So, for example,

Application.Workbooks refers to *all* the workbooks currently open in Excel.

Workbooks.Item(1) refers to the first workbook and is usually abbreviated to **Workbooks(1)**

Workbooks("Sales.xls") refers to the workbook by name.

A workbook usually contains worksheets, each of which contains ranges of cells. So, you might get cell B3 referred to as

Workbooks("Sales.xls").Worksheets("Sheet1").Range("B3")

That is a long description but fortunately it can usually be shortened. At any time, the workbook that you are working in is called the **active workbook**; the worksheet that is displayed is called the **active worksheet**. If you have more than one worksheet displayed on-screen, the worksheet that contains the cursor is the active one. If you have more than one workbook displayed on-screen, the workbook containing the active worksheet is the active workbook.

If you do not specify a particular workbook or worksheet, Visual Basic will use the active workbook and the active worksheet. If that is according to your wishes, then the long description above could be reduced to just **Range("B3")** as in the macro you recorded.

The **Sheets** collection contains *all* the sheets in a workbook, both chart sheets and worksheets.

Sheets("Year2001") refers to the sheet called **Year2001**.

Charts(1) refers to the first chart sheet on the tab bar.

5.2 Methods

Objects have methods that perform actions on them.

If you were considering the Range object, then examples of methods would be:

- Activate
- Clear
- Copy
- Cut
- Delete
- Select

The methods can be thought of as verbs (just as **bake** is a verb).

The syntax of many statements in Visual Basic is

Object.Method

which can be thought of as

Noun.Verb

rather like

Cake.Bake

and, in your macro,

Range("B3").Select

5.3 Properties

Each object has its own characteristics. In general, properties control the appearance of objects.

Thinking again about the Range object, typical properties would be

- ColumnWidth
- Font
- Formula
- Text
- Value

A property can be thought of as being somewhat similar to an adjective. It is set using a statement of the form

Object.Property = value

which can be thought of as

Noun.Adjective = value

rather like

Cake.Flavour = Chocolate

and, in your macro,

ActiveCell.FormulaR1C1 = "D.Morgan"

Each object has its own set of methods and properties.

An instruction such as

Range("C3").ColumnWidth = 14

sets the column width of cell **C3** to **14**. (Excel's default column width is 8.43 characters.) Since different characters take up different amounts of space this does not mean that you will necessarily get 14 characters in the cell.

When Range("C3").ColumnWidth appears on the left-hand side of the equals (=) sign, it is being given a new value (being written to).

When a property is on the right-hand side of an equals sign, you are reading from it. So,

Range("A1").ColumnWidth = Range("C3").ColumnWidth + 5

takes the value of the column width of cell **C3**, adds 5 to it, and then assigns that value to the column width of cell **A1**.

5.4 Variables

Just as in other programming languages, you can use variables. You do not have to declare variables; Visual Basic will automatically create storage for a variable the first time you use it.

Automatically created variables are of type Variant and can contain any type of data — strings, numbers, Boolean values, errors, arrays or objects.

For example, the following statement assigns the value 34 to the variable x.

```
x = 34
```

In the example below, variables **myl** and **myw** are given initial values. These are then used in a calculation.

```
myl = 34
myw = 15
myarea = myl*myw
```

5.5 Using Dim

If you need to specify what kind of data you are working with, you can declare the variable using a Dim statement of the form

Dim *variablename* As *datatype*

Possible data types are:

Boolean	True (1) or False (0)
Integer	-32,768 to 32,767
Long	-2,147,483,648 to 2,147,483,647
Single	-3.402823E28 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values
Double	-1.79769313486232E308 to –4.94065645841247E-324; 4.94065645841247E-324 to 1.79769313486232E308
Currency	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Date	January 1, 1900 to December 31, 9999
Object	Any object reference
String	Strings of length from 0 to about 65,535
Variant	Any numeric value up to the range of a Double, or any character text

If a variable exceeds the range of its data type you get an Overflow error. Error handling will be discussed in section 12.1.

An example of using Dim in a procedure is:

```
Sub example()
   Dim myint As Integer
   myint = 5
End Sub
```

A variable declared in a procedure is local to that procedure and other procedures cannot change its value. If you want other procedures to have access to that variable, declare it at the top of the module, before any Sub statements.

If you declare a variable at both module and procedure levels, the procedure-level variable is used within its procedure and the module-level variable is used in all other procedures.

In this next example, a variable **d** is given an initial value and then used to count how many cells in the range **A1:B10** hold values less than 40.

```
Dim  marks, c, d
Set marks = Range("A1:B10")
d = 0
For Each c in marks
   If c.Value < 40 Then
      d = d+1
   End If
Next c
```

5.6 Constants

Values that don't change should be set up as constants rather than variables. This prevents them being changed by accident.

The line

```
Const pi = 3.14159
```

will create the constant pi that can then be used in an expression such as

```
Rtangle = pi/2
```

5.7 Arrays

Arrays contain a sequence of variables. Each one is called an element of the array and is identified by an index number.

Dim can be used to declare an array without giving it any values.

Example 1

```
Dim myfriends(1 to 50) As String
```

creates a one-dimensional array that can contain 50 strings.

A typical statement in a procedure might then be

```
firstfriend = myfriends(1)
```

Example 2

```
Dim heightvage(1 to 80, 1 to 80) As Single
```

creates a two-dimensional array.

Example 3

The function called **Array** can create an array from a list of variables as in

```
Children = Array("Michael", "Bridget", "Peter")
```

When Array is used, the variables are of type Variant.

Note: **Redim** can be used to set up an array that can be re-sized but that will not be described in this document.

5.8 Using Set

Although most methods return values some, like Offset (see section 8.6), return objects. You cannot assign an object to a variable using an equals sign. Instead, you should use the Set statement, for example,

```
Set rangeOffset = Range("C1:C5").Offset(1,1)
```

6 Getting Help

6.1 Context-sensitive help

While you are writing a macro in the Microsoft Visual Basic window, you can access help about any particular item (such as **Range**) as follows:

- Select the item.
- Press **F1**.

Excel will display the appropriate help if it is available.

6.2 Help topics dialog box

To access the **Help Topics** dialog box,

- Make sure the Visual Basic window is the active one.
- From the **Help** menu, select **Microsoft Visual Basic Help**.
- You can either perform a search or choose whichever topic is most appropriate for your needs from the list.

6.3 Object Browser

The Object Browser can be used to view the objects, methods and properties of Excel in addition to many of the functions and statements provided by Visual Basic.

1. If Microsoft Visual Basic is already active, switch to that window (if not, select **Tools | Macro | Visual Basic Editor**).
2. From the **View** menu, select **Object Browser** (or press **F2**).

The scroll box in the middle contains a list of the various **classes** of objects.

A class is just a description of a type of object (for example, a *particular* chart belongs to the Chart class). Classes belong to a project or library.

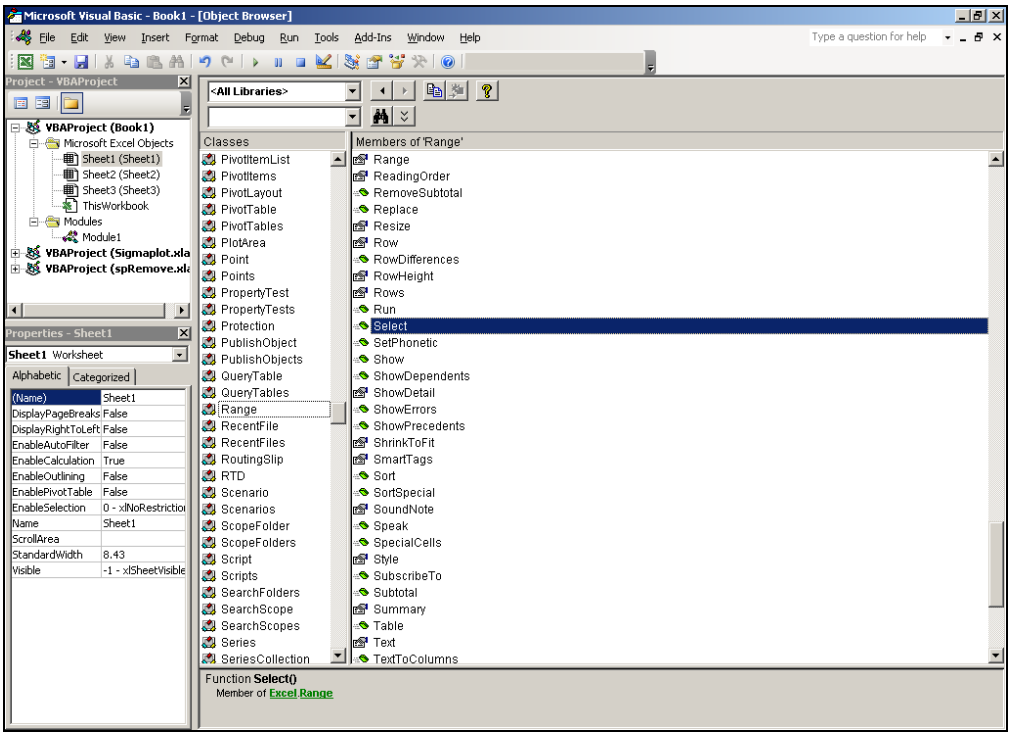
3 Click on the downward pointing triangle beside **<All Libraries>** and select **Excel**.

Now, just those classes belonging to Excel are displayed.

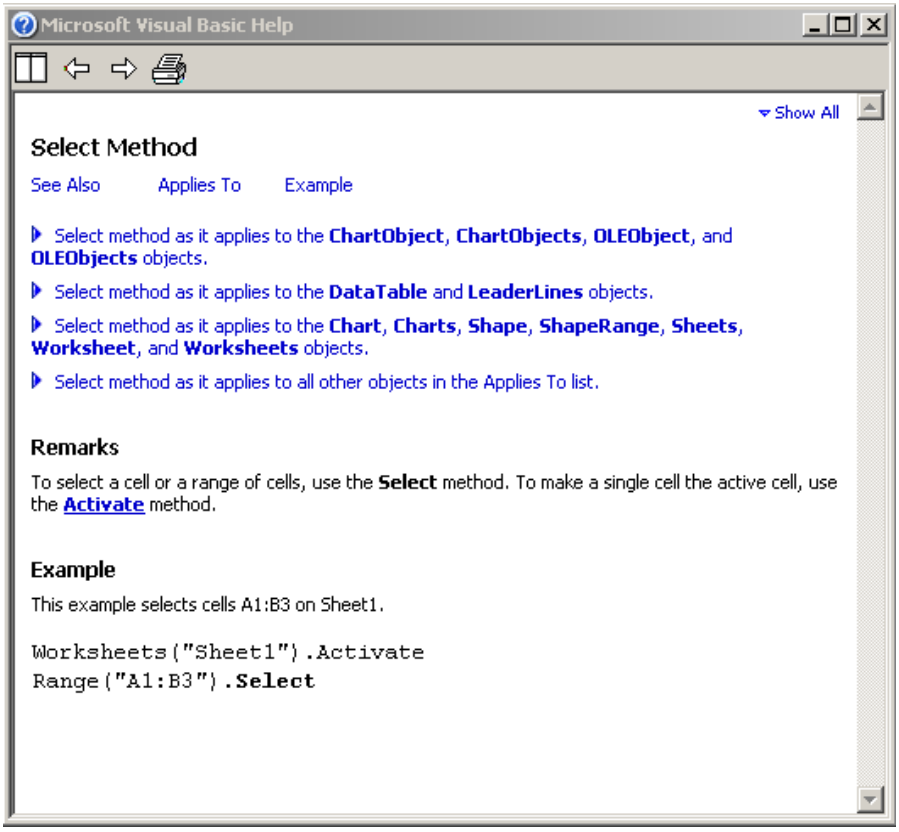
4 In the **Classes** area, scroll down until you can select **Range**.

5 In the **Members of 'Range'** area, scroll down and click on **Select**.

6 Note that beside **Select** there is a green symbol indicating that Select is a method.



7 To get more information about Select, click on the **Help** button — the one showing a yellow question mark.



8 Click on **Example** in blue.

9 The example given, **Range("A1:B3").Select** is similar to the line **Range("B3").Select** in your macro.

10 Close the window giving information about Select and scroll until you can see **Formula** in the **Members of 'Range'** box.

11 Note that this has a different symbol beside it — a pointing hand — indicating that Formula is a property.

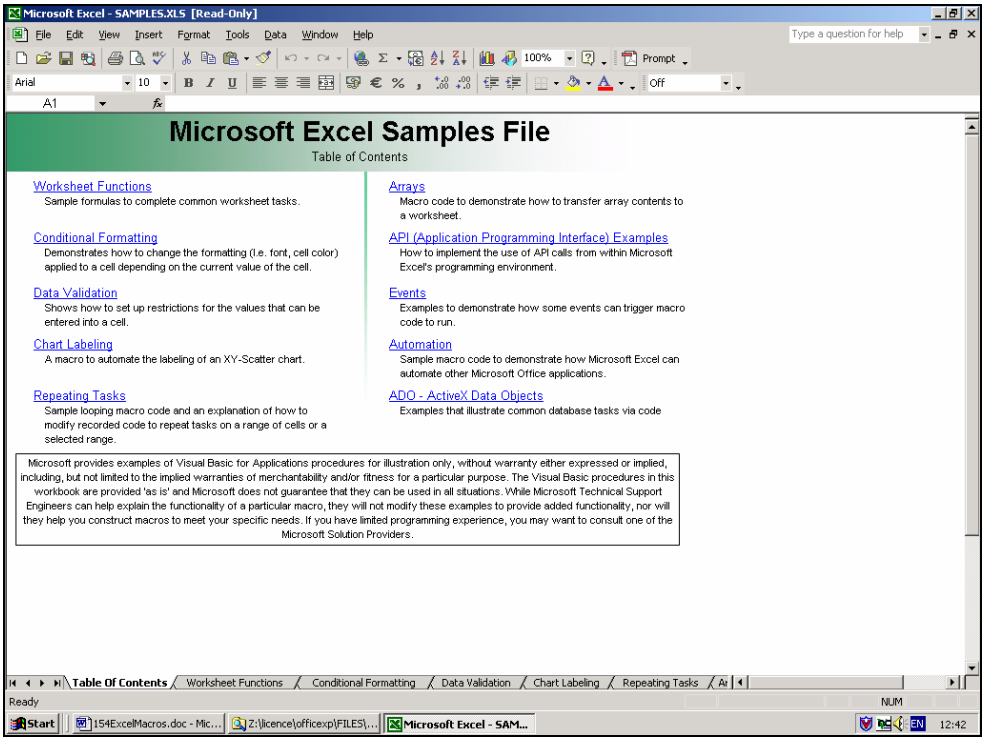
12 Close the **Visual Basic Object Browser** window.

6.4 Sample files

Excel used to provide some examples of macros in a file called **Samples.xls**.

Although the location can vary between different versions of Microsoft Office, on stand-alone machines, you should find this file in a folder similar to:
C:\Program Files\Microsoft Office\Office10\samples

This file is no longer part of the Office 2003 distribution, so will not be found on ITS Networked PC Service machines. The following screenshot was taken using Office 2002:



It can be very instructive to look at macros written by other people. You should however be critical; not everyone writes high-quality code.

7 Writing your own macros

When you recorded a macro, Excel created a module, added it to your workbook and wrote your recorded actions in a procedure belonging to that module.

When you want to write your own code in a new workbook *you* have to add a module to the workbook. The interface for macro development is called the Visual Basic Integrated Development Environment (IDE). Macro modules are displayed in the IDE instead of as a sheet in a workbook (as in versions before Excel 97).

7.1 Writing the macro

First move to a new workbook (but leave the old one open) as follows:

1 Click on the **New** button on the toolbar (or select **New** from the **File** menu and click **OK**).

Then give the name **Text** to **Sheet1** as follows:

2 Right-click on the tab of **Sheet1** and select **Rename**.

3 Type **Text** and press **Enter**.

To write the macro:

1 From the **Tools** menu, select **Macro** and then **Visual Basic Editor**.

2. From the **Insert** menu in the **Microsoft Visual Basic** window, select **Module**.
3. If you wish, you can change the name of this module. In the **Properties** window, beside **(Name)**, select the name **Module1,** and change it to **Experimenting**.
4. Click in the blank area of the **Experimenting (Code)** window.
5. Type

   Sub MyFirst

   and press **Enter**.

   Note how the **()** and **End Sub** are filled in automatically.
6. Type instructions to carry out the steps described below. You may find it helpful to refer to the listing of Address_abs() , shown in the screen dump in section 4, since your macro will contain similar instructions.

   Step 1: Select the sheet called **Text** (using Sheets("Text").Select)

   Step 2: Put the text I can write macros! in cell **B2** on that sheet

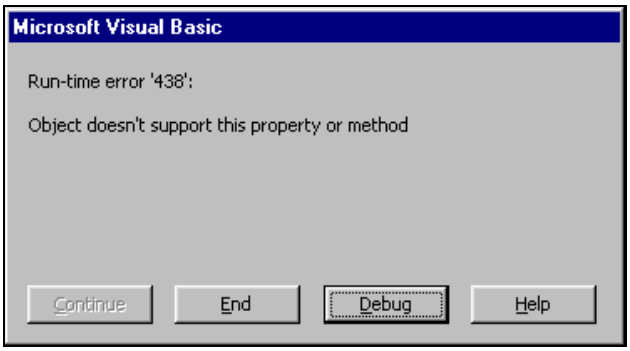
   Step 3: Make that text Bold

Finally, test your **MyFirst** macro:

1. Return to the sheet **Text**.
2. From the **Tools** menu, point to **Macro** and then select **Macros**.
3. In the **Macro** dialog box, select the macro called **MyFirst** and click **Run**.

All being well, the text **I can write macros!** will be entered on sheet **Text** in cell **B2**.

It is possible that you will get an error message. An example is shown below.



If you do have an error, proceed as follows:

1. Click on the **Debug** button and try to work out what the problem is.

The first faulty statement in your macro will be highlighted in yellow.

2 Edit the statement containing the error.

The yellow arrow in the margin will remain, indicating that you are in break mode.

3 *Either*
Click the **Reset** button
or
Move to Design mode by selecting **Run | Design Mode** (or click the **Design Mode** button).

4 Run the macro (press **F5**).

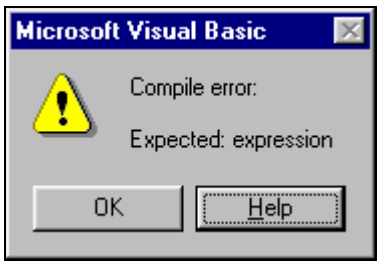
5 Return to the sheet **Text** to see whether the macro worked correctly.

7.2 Correcting mistakes

When you type a line in a macro and press the Enter key, Excel checks that line.

If it finds a term that it understands, like **range**, it will capitalise it to **Range**.

If it decides that information is missing or detects an error, it will make the line red and give you an error message.



If you want to get more information about the error, click on the **Help** button in this dialog box. To correct the error, click on **OK** and then amend the faulty line.

Sometimes there may be an error in your code that will not be apparent until Visual Basic tries to compile it before running the macro. In that case you will get a compile-time error. This will indicate the location of the problem and give you some idea of what is wrong.

Other errors may only become apparent when the macro is actually run. These are called run-time errors. To correct the error, click on the **Goto** button and amend the code.

Some “errors” however are not mistakes; they occur while the macro is running correctly. For example, a division by zero might happen unexpectedly. Your code should be written to cope with such situations using an **On Error** statement (see section 12.1).

7.3 Stepping through code

You can step through the code in a macro one line at a time. This helps you to see what each line does and can be very helpful if you are having difficulty in getting a macro to work as planned.

1 Move to a blank sheet in your workbook.

2 Click in cell **A1**.

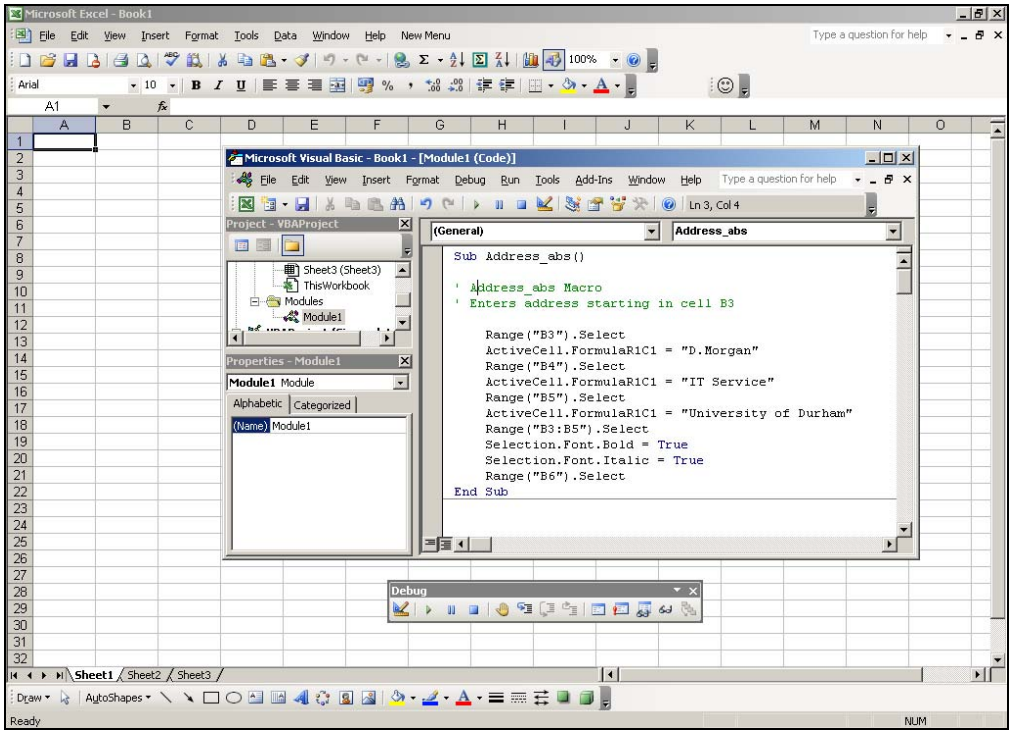
In order to experiment with your macro **Address_rel** (to be found in Book1-Module1), display it as follows:

3 *Either*
In the Excel window, select **Tools** | **Macro** | **Macros**, select the **Address_rel** macro name and click **Edit**
or
Double-click on **Module1** of **Book1** in the **Project-VBAProject** pane of the **Visual Basic** window.

Next, from within the Visual Basic window, display the **Debug** toolbar.

4 From the **View** menu, point to **Toolbars** and select **Debug**.

5 Make the **Visual Basic** window smaller so that you have a layout similar to the one shown below.



You need to be able to see both your worksheet and your code at the same time in order to be aware of what is happening to your worksheet as you step through the code.

6 Click at the beginning of your **Address_rel** macro.

7 Click the **Step Into** button on the **Debug** toolbar (or press **F8**).

8 Keep clicking the **Step Into** button (or press **F8**) to run the subsequent lines of code and watch your address appearing in your worksheet.

As each line is executed, Excel carries out the appropriate action and highlights the next line of code.

If there is an error, Excel stops and displays an error dialog box.

The **Step Over** button (equivalent to pressing **Shift+F8**) is similar to Step Into but differs in the way it handles a statement containing a call to a procedure. Whereas Step Into will work its way through the statements in the called procedure, Step Over treats the called procedure as a unit and steps to the next statement in the current procedure.

Step Out (equivalent to pressing **Ctrl+Shift+F8**) executes the remaining lines in the current procedure.

7.4 Immediate pane

You can experiment with a line of code, such as

```
ActiveCell.Value = "testing"
```

by typing it in to an **Immediate** pane. When you press the **Enter** key, Excel carries out that instruction. Once you have got the line right, you can copy it to the Clipboard and paste the line in to your macro. To access the **Immediate** pane, click on the **Immediate Window** button on the **Debug** toolbar (or press **Ctrl+G**).

7.5 Watches pane

You can keep an eye on the values of any variables that are in your code by using the **Watches** pane.

Step through your macro until you reach the point where the variable you are interested in is used. Select the variable in the Code pane and click the **Quick Watch** button (or press **Shift+F9**). The **Quick Watch** dialog box is displayed and shows the value of the variable. Click **Add** if you want to add that variable to the **Watches** pane. Then that, and any other nominated variables, can be watched by clicking the **Watch Window** button on the **Debug** toolbar.

7.6 Breakpoints

On some occasions, you may wish to run your macro but have it stop at some point before its natural end. This can be achieved by inserting a break point. Move your cursor to the line of code where you want to stop. Click the **Toggle Breakpoint** button on the **Debug** toolbar (or press **F9**). The line will then be highlighted in brown. When the macro is run, Excel will stop at that breakpoint.

To remove a breakpoint, move the cursor to the line containing the breakpoint and click the **Toggle Breakpoint** button (or press **F9**).

Note: Break points can only be set on lines of executable code (not on comments, for example).

7.7 Restarting and ending

Once your code has been stopped, by a break point or by stepping through a macro, you can:

- *resume* execution of the code by clicking the **Continue** button (or by pressing **F5**)
- *end* the macro, without resuming execution, by clicking the **Stop** button (or by closing the Debug window — double-click its Control Box)

7.8 Break mode and design time

In **break mode** the execution of your macro is temporarily suspended. You can look at your code, debug it, step through it or continue running it.

You enter break mode when

- a **Stop** statement or un-trapped run-time error is met during the execution of your macro
- a breakpoint is encountered in your macro
- you select **Break** from the **Run** menu (equivalent to pressing **Ctrl+Break**) or click the **Break** button on the **Debug** toolbar.
- you add a **Break When True** or **Break When Changed** watch expression (not discussed in this Guide)

While building an application you are in **design time**.

You enter **design time** when

- an **End** statement is reached in your code
- you stop execution manually by selecting **Reset** from the **Run** menu (or click the **Reset** button on the **Debug** toolbar)

8 Referencing cells and ranges

At this stage, it would probably be useful to look at some of the ways of referring to cells in a macro. If you are manipulating data on a worksheet, you will need to use the **Range** object. This can be either a single cell or a range of cells and is probably the most used object in Excel. Several ways of returning a Range object are described below.

8.1 A1 reference style

The following table gives examples of ways in which to refer to a cell, or range of cells, in the A1 reference style using the Range method.

Reference	**Refers to:**
Range("B1")	Cell B1
Range("B1:C6")	Range B1 to C6

Range("B1:D7, F8:J20")	Two areas of cells
Range("C:C")	Column C
Range("7:7")	Row seven
Range("B:D")	Columns B to D
Range("2:6")	Rows two to six
Range("2:2, 5:5, 9:9")	Rows two, five and nine
Range("B:B, D:D, G:G")	Columns B, D, and G

Example 1

Range("A1:B3").Select selects the range A1:B3

Example 2

This procedure gives an italic format to the cells **M2:P7** on a worksheet called **Spring**, in a workbook called **Year1**.

```
Sub  MakeItalic()
   Workbooks("Year1").Sheets("Spring").Range("M2:P7") _
      .Font.Italic = True
End Sub
```

Shortcut notation

A particular range of cells within square brackets can be used as a shortcut notation. Comparing with Example 1 above,

[A1:B3].Select also selects the range A1:B3

8.2 Index numbers

The **Cells** property can be used to return a Range object that represents a *single* cell. The cell's row and column index numbers have to be supplied in the form **Cells(*row_no, col_no*).**

If the index numbers are omitted, as in **Cells**, a Range object is returned that represents *all* the cells on the sheet.

Example 1

Cells(4,1) returns the cell **A4**

Example 2

```
Worksheets("Sheet2").Cells(3, 2).Value = 2000
```

enters the number **2000** in cell **B3** on **Sheet2** of the active workbook.

Example 3

```
Worksheets("Sheet2").Cells.ClearContents
```

clears the contents of all the cells on Sheet2 of the active workbook.

Note: The Cells property is very useful when writing a macro to loop through a range of cells.

8.3 Rows and Columns

There are a couple of properties called **Rows** and **Columns** that enable you to work with entire rows or columns.

Reference	**Refers to:**
Rows(4)	Row 4
Rows	All the rows on active worksheet
Columns(4)	Column D
Columns("D")	Column D
Columns	All the columns on active worksheet

Example

```
Worksheets("Week4").Rows(2).Font.Bold = True
```

gives a bold format to the contents of all the cells in the second row of the sheet **Week4** of the active workbook.

Note: It is possible to work with several rows or columns at once by using the **Union** method (not discussed in this document).

8.4 Named ranges

Sometimes it is easier to deal with a range when it has a name rather than have to use A1 notation. A name can be given to the range before the macro is written, or, the macro can assign the name to the range.

8.4.1 Name given to range outside the macro

To give a name to a range, select the range of cells, click in the **Name** box at the left-hand end of the formula bar, type a name and press the **Enter** key.

Suppose the name **Week2** has already been given to the cells **F12:F18** on the sheet **June** in the workbook **Year2000**.

Example 1

```
Range("[Year2000.xls]June!Week2").Font.Bold = True
```

will give a bold format to the cells in the named range **Week2**.

Example 2

If **Year2000** is the active workbook and **June** is the active worksheet, then

```
Range("Week2").Font.Bold = True
```

will give a bold format to the cells in the named range **Week2**.

8.4.2 Name given to range as part of the macro

A name can be assigned to a range from within a macro, as in the example below:

```
Workbooks("Year2000.xls").Names.Add _
   Name:="Week2", _
   RefersTo:="=June!D1:D10"
```

and then the instruction

```
Range("Week2").Font.Bold = True
```

will work as before.

8.5 Multiple ranges

It can be useful to refer to multiple ranges within a macro, perhaps to clear the contents of those cells.

```
Worksheets("June").Range("F12:F18, H12:H18, J12:J18").ClearContents
```

will clear the nominated cells on the worksheet June.

For named ranges on the same active worksheet, you could use

```
Range("Week2, Week3, Week4").ClearContents
```

Note: In a macro, multiple ranges can be defined, named, and then combined using the **Union** method.

8.6 Offset cells

The Offset property can be used to refer to a cell by stating where that cell is in relation to the active cell.

The general form is

Offset(*no_rows_down*, *no_cols_to_right*)

A negative value for *no_rows_down* is interpreted as meaning rows *up.*

A negative value for *no_cols_to_right* means columns to the *left.*

Example 1

Suppose that **B8** is the active cell. The following macro statement would give a bold format to the contents of cell **D17** (which is *nine* rows down from **B8** and *two* columns to the right):

```
ActiveCell.Offset(9, 2).Font.Bold = True
```

Example 2

```
ActiveCell.Offset(1,-2).Select
```

selects the cell which is one row down and two columns to the left of the active cell. That new cell becomes the active one.

Example 3

```
ActiveCell.Offset(0,1).Value = "whatever"
```

puts an entry in the cell which is in the same row but one column to the right of the selected cell.

8.7 R1C1 reference style

When using R1C1 reference style, Excel refers to cells by their row and column *numbers*. As an example, the cell reference **R4C2** refers to cell **B4**.

(Should you ever wish to see this in action on your worksheet, rather than in a macro, select **Options** from the **Tools** menu, click on the **General** tab and select the **R1C1** option.)

Using this notation, relative cell references in a calculation are expressed in relation to the cell containing the formula.

R[m]C[n] refers to the cell *m* rows down and *n* columns to the right of the active cell. The values of *m* and *n* can be negative (in which case they mean up and left, respectively).

Example: Enter the formula SUM(B2:B4) in cell B5

The content of the cell in row 5 column B is to be the sum of the contents of the cells in the rows three before, two before and one before that fifth row, but in the same column.

This translates into the macro instructions

```
Range("B5").Select
ActiveCell.FormulaR1C1 = "=SUM(R[-3]C:R[-1]C)"
```

Example: Enter the formula F2-F4 in cell D5

The macro instructions for this could be

```
Range("D5").Select
ActiveCell.FormulaR1C1 = "=R[-3]C[2]-R[-1]C[2]"
```

The R1C1 in FormulaR1C1 can be omitted and, if you wish, the two lines can be combined into

```
Range("D5").Formula = "=R[-3]C[2]-R[-1]C[2]"
```

Example: Change a formula to its resulting value

The following macro instructions will put the formula **=G5*G4** in cell **G6** and then change the contents of **G6** from the formula to the resulting value (the answer).

```
Range("G6").Select
ActiveCell.Formula = "=R[-1]C*R[-2]C"
Selection.Copy
Selection.PasteSpecial  Paste:=xlValues
Application.CutCopyMode = False
```

The final statement cancels the Cut/Copy mode status so that the marquee around the selected cells is removed.

8.8 Exercises

Before trying these exercises:

1. Close any open workbooks and start a new one.
2. Enter the following data in cells **B4:F9** on **Sheet1**. Use AutoFill to help you to do this quickly.

X	X	X	X	X
X	Mon	Tue	Wed	X
Jan	Feb	Mar	Apr	May
X				X
X				X
X	X	X	X	X

3. Copy that data to the corresponding cells on **Sheet2** and **Sheet3** so that you have spare copies on which to try out your macros.
4. Activate **Sheet1**.

Exercise 1

Write a macro called **DeleteValues** that uses **Range** (see section 8.1) and the **ClearContents** method to delete the values (Mon, Tue, Wed) in cells **C5:E5**. Run the macro.

Exercise 2

Write a macro called **EnterValue** that uses **Cells** (see section 8.2) to enter the value **Year 2000** in cell **D7**. Run the macro.

Exercise 3

Write a macro called **DeleteRow** that uses **Rows** (see section 8.3) and the **Delete** method to delete row **6** (the row containing Jan, Feb, …). Run the macro.

When you have completed these exercises, the final result should be

X	X	X	X	X
X				X
X		Year 2000		X
X				X
X	X	X	X	X

Solutions can be found in section 16.2.

9 Decisions

Sometimes, you might want your macro to test for specific conditions on your worksheet and then respond accordingly. There is an **If…Else…End If** sequence to help with such situations. Control structures like this cannot be recorded, you have to write them in Visual Basic.

The **If** Visual Basic keyword is analogous to the **IF** worksheet function.

9.1 IF statement

The simplest, **one line**, form of an **If** statement is

If *condition* Then *statement1* [Else *statement2*]

In this notation, anything in square brackets [ ] is optional — it can be omitted if you do not need it.

If the *condition* is satisfied (true) then *statement1* is carried out otherwise control passes to *statement2*.

Usually you will need to use the **block** statement **If…Then…Else** which does not restrict you to one line of code. This has the form

```
If condition Then
   one or more VB statements for action1
[ElseIf condition2
   one or more VB statements for action2 ...]
[ElseIf condition3
   one or more VB statements for action3 ...]
[Else
   one or more VB statements for action4   ]
End  If
```

In the general form above, the ElseIf and Else are optional (as denoted by the square brackets). Only two ElseIf are shown above; you can have more if you wish.

Example

This macro looks at the value in **A1**. If it is 100 it enters the text **Full marks** in **B1**. If the value in **A1** is not 100, the macro enters **No** in **B1**.

```
Sub fullmarks()
   Sheets("Sheet1").Select
   Cells(1,1).Select
   If ActiveCell = 100 Then
      Cells(1,2).Value = "Full marks"
   Else
      Cells(1,2).Value = "No"
   End If
End Sub
```

Note: You could omit the line **Cells(1,1).Select** and change the following line to **If Cells(1,1) = 100 Then**

Exercise: Using If Then Else

1. On a blank worksheet, enter the data shown below into cells **F4:G5.**

Marks gained	Result
57	

2. Write a macro called **PassFail** which puts the text **Pass** in **G5** when the mark gained is greater than **49**, but enters **Fail** otherwise.
3. Run the macro. Did you get the result **Pass**?
4. Alter the value in **F5** to **39** and run the macro again. Did you get the result **Fail**?

(Solution in section 16.2.)

9.1.1 Using an IF statement to end a macro

The statement

```
If ActiveCell = "" Then End
```

can be used as a way of ending a macro.

Example

Suppose you have a macro that is looking at a cell, and you want it to

- stop if that cell is blank
- enter the text **good** in the neighbouring cell to the right, when the value in the inspected cell is greater than or equal to 40
- enter the text **poor** in the neighbouring cell to the right, when the value in the inspected cell is *not* greater than or equal to 40

The following code would achieve that purpose:

```
If ActiveCell = "" Then End
If ActiveCell >= 40 Then
                    ActiveCell.Offset(0,1).Value = "good"
Else
                    ActiveCell.Offset(0,1).Value = "poor"
End If
```

9.2 Select Case

Select Case is similar to **If…Then…Else**, except that you can choose from several condition values.

The general form is

```
Select Case expression
   Case value1
      one or more VB statements for action1
   Case value2
      one or more VB statements for action2
   … … …
   Case valuem
      one or more VB statements for actionm
    [Case Else
      one or more statements for action otherwise]
End Select
```

When the macro is running, Visual Basic takes the value of the *expression*, matches it against those values given in the Case statements and executes the appropriate lines of code. If there isn't a match, control passes to the Case Else and its lines of code are run.

You can have as many **Case** lines as you like and you can have more than one value on each line. You can even have a range of values by using **To**.

Example

In the following example, the variable n takes as its value the content of the active cell. Control then passes to the corresponding Case statement.

Note the use of **To** in one of the Case statements.

```
n = ActiveCell
Select Case n
Case 12
   statementA
Case 24, 36
   statementB
Case 48 To 96
   statementC
… …
Case Else
   statementD
End Select
```

Exercise: Using Select Case

1 On your worksheet, enter the data shown below into cells **R2:S3**

Month	Time of year
6	

2 Write a macro called **SSAW** (representing **S**pring, **S**ummer, **A**utumn, **W**inter) that puts, in cell **S3,**

Spring	for months **3 to 5**
Summer	for months **6 to 8**
Autumn	for months **9 to 11**
Winter	for months **12**, **1** and **2**

3 Run the macro. The **Time of year** entry should be **Summer**.

4 Change the value in **R3** to **12** and run the macro again. Is the time of year correct?

5 Try another month. (Solution in section 16.2)

9.3 Constructing conditions

Conditions can become quite complicated.

In such situations, **And** and **Or** can help to make them elegant and more easily understood.

9.3.1 Use of AND

In the example

```
If (condition1) And (condition2) Then
   statement1
Else
   statement2
End If
```

statement1 will only be executed if *both* ***condition1*** and ***condition2*** are true.

In all other situations statement2 will be executed.

In situations where the logic is unambiguous, the brackets around the conditions may be removed.

Exercise: Using IF with AND

1 On your worksheet, enter the data shown below into the cells **W2:X3**

Age	Send leaflet?
19	

A company wishes to send leaflets to anyone in the age range 21 to 35 inclusive.

2 Write a macro called **Leaflet** that enters **Yes** in **X3** when the age in **W3** is of interest to the company, and **No** otherwise.

3 Run the macro. Did you get the answer **No**?

4 Change the value in **W3** to **30** and run the macro again. Did you get the answer **Yes**?

5 Change the value in **W3** to **50** and run the macro again. Did you get the answer **No**? (Solution in section 16.2)

9.3.2 Use of OR

In the example

```
If (condition1) Or (condition2) Then
   statement1
Else
   statement2
End If
```

statement1 will be executed if *at least one* of ***condition1*** and ***condition2*** is true.

If neither ***condition1*** nor ***condition2*** is true, statement2 will be executed.

9.3.3 Use of multiple AND and multiple OR

The example shown next contains a straightforward extension of the use of **And** as described in section 9.3.1.

Note the use of an underscore as the continuation character.

```
If (condition1) And (condition2) _
   And (condition3) And (condition4) Then
   statement1
End If
```

All four conditions would have to be true for *statement1* to be executed.

Multiple **Or** can be used in a similar way.

9.4 Line labels and numbers

It can sometimes be useful to label a line of a macro. Each line label must be unique within the module and end with a colon. The label can start anywhere on a separate line provided that the label is the first non-blank character on the line.

A line number works in the same way but uses numbers instead of characters.

Look back at section 9.1.1 to remind yourself what the code there did. In the example below, a line label, Offset and a GoTo are used to work through a range of cells until an empty one is reached.

```
begin:
   If ActiveCell = "" Then End
   If ActiveCell >= 40 Then
      ActiveCell.Offset(0,1).Value = "good"
   Else
      ActiveCell.Offset(0,1).Value = "poor"
   End If
   ActiveCell.Offset(1,0).Select
GoTo begin
```

10 Passing information

There are a couple of easy ways of passing information between you and the macro while it is running.

10.1 Message box

The *statement*

MsgBox *variable*

displays, on your screen, the current value of *variable* in your macro. This can be very useful while you are debugging a macro.

The MsgBox *function* has the general form

MsgBox(*prompt* [, *buttons*] [, *title*] [, *helpfile*, *context*])

It displays a message in a dialog box, waits for you to click a button, and then returns an integer indicating which button you clicked.

prompt is essential and determines the message shown

buttons is optional and controls the type of buttons shown (OK, Cancel etc)

title is optional and sets the text in the title bar of the dialog box

helpfile is optional and controls which Help file is to be used. If *helpfile* is included, a value for *context* must also be provided.

context is optional and is the Help context number. If *context* is included, a value for *helpfile* must also be provided.

For a full description of this function, consult Visual Basic's in-built help.

10.2 InputBox

The InputBox function displays a prompt in a dialog box, waits for the user to input text or click a button, and then returns a string. It is an ideal way of getting a single value or cell address from you to the macro while it is running.

Its general form is

InputBox(*prompt* [, *title*] [, *default*] [, xpos] [, *ypos*] [, *helpfile*, *context*])

where

prompt is essential and determines the message shown

title is optional and sets the text in the title bar of the dialog box

default is an optional default response (if omitted, the text box is empty)

xpos is an optional setting for the distance of the left edge of the dialog box from the left edge of the screen (if omitted, the box is centred horizontally). It is measured in twips.

ypos is an optional setting for the distance of the upper edge of the dialog box from the top of the screen (if omitted, the box is

positioned about one-third of the way down). It is also measured in twips.

helpfile is optional and controls which Help file is to be used. If *helpfile* is included, a value for *context* must also be provided.

context is optional and is the Help context number. If *context* is included, a value for *helpfile* must also be provided.

A ***twip*** is one twentieth of a point. There are 72 points in an inch.

If values for more than just *prompt* are included, InputBox must be part of an expression.

10.3 Examples of macro instructions

Example 1

MsgBox n will display on screen the current value of n in the macro.

Example 2

MsgBox "Tax to be paid is " & TotalTax will display something like

Tax to be paid is £45.15

Example 3

```
myAnswer = MsgBox ("Reached end of first data set. Continue? " , _
                                    vbOKCancel, "Analysing data")
```

will produce the following message box

When one of the buttons is clicked, the message box closes and a corresponding value is returned to the macro. This value can then be used, as in

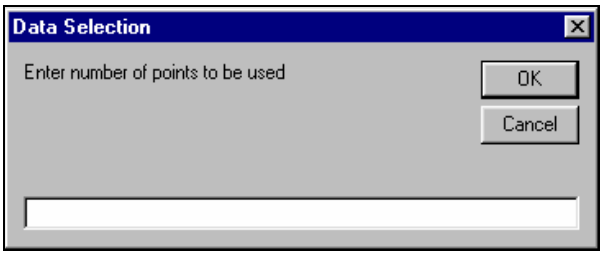
```
If  myAnswer = vbCancel  Then  Exit Sub
```

Example 4

The line of code

```
nPoints = InputBox ("Enter number of points to be used" , _
                                    "Data Selection")
```

will produce the following message box



It is important to note that this function returns a *string* containing what you typed. Most of the time Visual Basic will not mind if you use the response as though it were a number (as in the example below).

```
nPoints = InputBox("Enter number of points to be used", _
            "Data selection")
total = 0
For x = 1 To nPoints      ' The For … Next x
   total = total + x      ' structure is discussed
Next x                                    ' in Section 11
MsgBox total
```

Should it be necessary, you can convert the text to a number using the VAL function as in **n= VAL(nPoints)**

Note: Exercises in section 11 will give you the chance to use **MsgBox** in a macro.

11 Repeating actions — loops

Loops and If statements are known as control structures. A loop allows a group of statements to be run several times.

Three kinds of loop will be considered at this stage:

- Do
- For Next
- For Each Next

11.1 Do

This can be used to run a block of statements an indefinite number of times. A condition can be evaluated inside the loop to decide whether to continue running or not. Its general form is:

```
Do
   various statements
Loop
```

Example

```
m = 4
Do
   m = m + 1
   MsgBox  m
   If  m >6 Then Exit Do
Loop
```

In this example,

- m is given the value 4
- inside the loop, m is increased to 5
- a message is shown on screen consisting of the value of m, 5
- m>6 is FALSE (since m is only 5) so the loop continues
- m is increased to 6
- a message 6 is shown on screen
- m>6 is FALSE (since m is only 6) so the loop continues
- m is increased to 7
- a message 7 is shown on screen
- m>6 is TRUE (since m is 7) so the Exit Do is carried out and the loop terminates

11.2 Do While

The Do While loop continues as long as a condition is true. Its most general form is given below. Anything in square brackets is optional.

The **|** inside the first set of square brackets indicates that either While *condition1* or Until *condition2* could be used.

```
Do  [While  condition1 | Until condition2]
   statement1
   statement2
Loop [While  condition3 | Until condition4]
```

The loop executes while a condition is true or until it becomes true. You can have a test condition

- at the top of the loop
- at the bottom of the loop (the statements in the loop will be executed at least once)
- not at all (known as an infinite loop since the statements would repeat for ever)

If you ever need to break out of an infinite loop, press **Ctrl/Break**. The Macro Error dialog box will be shown and you can click on **End** to end your procedure.

Example

```
m = 4
Do  While  m < 7
   m = m + 1
   MsgBox  m
Loop
```

In this example,

- m is given the value 4
- m<7 is TRUE so the loop is entered
- m is increased to 5
- a message 5 is shown on screen
- control passes to the beginning of the loop
- m<7 is TRUE (since m is only 5) so the loop continues
- m is increased to 6
- a message 6 is shown on screen
- control passes to the beginning of the loop
- m<7 is TRUE (since m is only 6) so the loop continues
- m is increased to 7
- a message 7 is shown on screen
- control passes to the beginning of the loop
- m<7 is now FALSE (since m is 7) so the loop terminates

Exercise: Using a Do While or Do Until loop (hints at end of exercise)

1. Activate a blank worksheet .
2. Enter the values shown, including the **End of data**, into cells **G3:G9**.
3. Write a macro called **AddNumbers** to add up these numbers as follows:
   - initialise a variable called **total** to zero
   - add each number in turn to the total
   - keep going round the loop until the **End of data** entry is reached

12
23
34
67
78
89
End of data

4. Outside the loop, use **MsgBox** to display the answer.
5. Run the macro. The answer should be 303.
6. Insert two more rows of data (before the **End of data** row), containing values **45** and **56**.
7. Run the macro again. The answer should be 404.

Hints: You could have a variable **r** representing the row number of the cell you are dealing with, and initialise this to be **3**. CELLS(r , 7) will then refer to **G3** (the first value). Each time round your loop, increase r (using the

statement r = r + 1) so that you deal with G4, G5, and so on, in turn. (Solution in section 16.2)

11.3 For Next loop

This kind of loop is used when you want to repeat an action a specified number of times rather than when a condition is met. A counter variable increases (or decreases) in value for each repetition of the loop.

General form:

```
For counter = firstvalue To lastvalue [Step increment]
   statement1
   statement2
Next [counter]
```

Your code is easier to read if you use **Next *counter*** rather than just **Next**.

The value of the counter can be used in the statements. So, statement1 could be something like **Cells(*counter*,1).Value = *counter* + 50**

Example

The following code will add up the numbers from 5 to 50, using a variable called total (initialised to zero).

```
total = 0
For index = 5 To 50
   total = total + index
Next index
```

If you want to experiment with putting that code into a macro and running it, the statement

```
MsgBox total
```

will display the result for you.

Increments other than 1

Increments can be greater than 1 in order to skip values. In the example below, x takes the values 1, 3, 5, and so on up to and including 15.

```
For x = 1 To 15 Step 2
   statement1
   statement2
Next x
```

In order to count backwards, negative increments can be chosen:

```
For x = 100 To 5 Step -5
   statement1
   statement2
Next x
```

Here, x takes the values 100, 95, 90, and so on down to 5.

Note: **If** structures can be put within a loop.

Exercise: Using a For Next loop

1 Activate a new worksheet.

2 Enter the data shown into cells **B20:C26**.

Surname	First Name
Astikoff	Ivan
Beach	Sandy
O'Shea	Rick
Dover	Eileen
Teak	Anne
Mann	Andy

3 Write a macro called **Names** to:

- put the text **Full Name,** with a Bold format, in cell **D20**
- for each person in turn, construct their full name (for example, Ivan Astikoff) and put it in the correct row of the **D** column

4 Run the macro. (Solution in section 16.2)

11.4 For Each Next loop

This is similar to a **For Next** loop but instead of repeating the statements a specific number of times, they are repeated for each element in a collection of objects or in an array.

General form:

For Each *element* In [*array* | *collection*]
one or more Visual basic statements
Next [*element*]

In the example below, the selected range object is assigned to the variable **aname** and **c** is a cell. Each cell in the range **A1:B10** has 10 added to it.

```
Dim aname, c
Range("A1:B10").Select
Set aname = Selection
For Each c in aname
   c.Value = c.Value +10
Next c
```

11.5 Exit statement

You should be able to set up your loops so that they come to a graceful conclusion. Just occasionally you may need to get out abruptly from an **If** or **Select Case** statement that is inside a loop.

An **Exit Do** statement (to get out of a Do loop) or an **Exit For** (to get out of a For loop) can be used on these, hopefully rare, occasions.

Example

```
For  x = 1 To 20
   If condition1 Then Exit For
   If condition2 Then
      statement1
   Else
      Statement2
   End If
Next x
```

11.6 Nested loops

Loops can be nested. The following example involves just two loops:

```
For x = 1 To 10
   For y = 1 To 30
      statement1
   Next y
   Statement2
Next x
```

The flow of control is as follows:

- x takes the value 1
- y takes the value 1, statement1 is executed
- y is increased to 2, statement1 is executed
- …….y keeps increasing until…….
- y is increased to 30, statement1 is executed
- statement2 is executed
- x is increased to 2
- y takes the value 1, statement1 is executed
- etc

12 Determining the extent of data

There may be occasions when you do not know how many cells will be used to store your data. So, you need to determine the extent of the data from within the macro itself.

A region of cells is a block of cells surrounded by blank rows and columns.

***range*.End (*Direction*)**	returns the cell marking the extreme boundary of a region of used cells.

Direction can be **xlToLeft, xlToRight, xlUp,** or **xlDown**.

***range*.Row**	returns the row number of the first cell in that range.
***range*.Column**	returns the column number of the first column in that range.

Some examples may help to clarify this.

Example 1

The instruction

```
ActiveCell.End(xlDown).Select
```

selects the end of the block, moving down from the active cell.

Example 2

The instruction

```
n = ActiveCell.Row
```

sets **n** equal to the row number of the active cell.

Example 3

Suppose you want to add up a column of numbers on a worksheet but you do not know how many numbers there are.

The first data value is in **B4** on **Sheet1** and the other numbers to be added are below that, also in column **B**. This macro will calculate the sum (in the variable **d**) and put that value in the cell below the numbers. A border is then put around the cell containing the total.

```
Sub add()
   Sheets("Sheet1").Select
   Cells(4,2).Select
   ActiveCell.End(xlDown).Select
   n = ActiveCell.Row
   d = 0
   For x = 4 To n
      d = d + Cells(x,2).Value
   Next x
   Cells(n+1,2).Value = d
   Cells(n+1,2).BorderAround  Weight:=xlMedium
End Sub
```

Exercise: Dealing with a list of unspecified length

Before starting this exercise you should copy a file from the **T:** drive of the Networked PC service to your own file space. This will provide you with the data needed.

The file is **T:\its\Excel\MacroData.xls**

If you are using a stand-alone PC, you may like to get a copy of this from the ITS WWW pages under **Information | Guides | Sample Files** (**http://www.dur.ac.uk/its/info/guides/files/excel/**)

1. Open your copy of the file called **MacroData.xls**
2. Select the **Names** sheet.
3. Write a macro called **LongList** to fill in the full names in column C. Structure the macro so that it will work for any number of names (a

blank row denotes the end of the list).
You may like to refer back to the macro **Names** in section 11.3.

4 Run the macro and check that it has worked correctly.

5 Add some more names and run the macro again.

6 Check that it has worked correctly. (Solution in section 16.2)

12.1 Macros using pre-selected data

Sometimes you may need to write a macro that uses data from whichever cells are selected before the macro is run.

In that case, **Selection.Cells(1,1)** refers to the top-left cell of the *selection* (rather than the worksheet) and all other Cells references will be relative to that one. For example**, Selection.Cells(2,1)** will be the first cell in the second row of your selection.

In a macro,

Selection.Rows.Count will give the number of rows in your selection

Selection.Columns.Count will give the number of columns

The following example will enable you to see how these instructions can be used.

Example: Re-scoring a questionnaire

Suppose the numbers 1 to 5 have been used to code the answers from a questionnaire and that blank cells correspond to missing answers.

The following macro will re-score 5 to 1, 4 to 2, 3 to 3, 2 to 4 and 1 to 5 while leaving blank cells unchanged.

The area of cells to be re-coded must be selected before the macro is run.

```
Sub Rescore()
' Select area of cells to be re-scored before running this macro
' Numbers are changed 5 to 1, 4 to 2 etc, blank cells stay blank
   nor = Selection.Rows.Count
   noc = Selection.Columns.Count
   MsgBox "There are " & nor & " rows and " & _
               noc & " columns of data"
   For i = 1 To nor
      For j = 1 To noc
         score = Selection.Cells(i, j)
         If score <> 0 Then
            Selection.Cells(i, j) = 6 – score
         End If
      Next j
   Next i
End Sub
```

13 Error handling

An error may occur while your macro is running. This could be because you have divided by the number in a cell and that number happened to be zero. Whatever the cause, you need to deal with these situations.

As an example, suppose you have the following data in cells **B4:C8** on **Sheet4**.

123	2
234	3
345	0
456	4
567	5

The first version of a macro to divide the numbers in column **B** by their adjacent values in column **C** and store the results in column **D** could well be as shown below.

```
Sub ErrorTestingVersion1()
   For r = 4 To 8
      Cells(r,4) = Cells(r,2) / Cells(r,3)
   Next r
End Sub
```

This, however, would generate the error message

Run-time error '11':
Division by zero

because of division by the zero in **C6**.

You can tell Excel what to do when an error occurs by giving instructions within the macro. Three approaches are described in the following sections.

13.1 To deal with an error

One way is to include the statement

```
On Error GoTo errorchk
```

at the beginning of your macro with

```
GoTo fin
errorchk:   MsgBox "Error,recheck data"
fin:
```

outside the loop, just before End Sub. When the division by zero is encountered, the loop will terminate in a controlled fashion and a helpful message will be shown on screen. The calculations up to that point will have been done but the rest will not be attempted.

13.2 To ignore an error

Another, better, way is to include the statement

On Error GoTo Last

in your macro, where Last is a label attached to the **Next r**

Last: Next r

The effect of this will be to continue doing calculations and just omit the result for **C6**.

13.3 To continue with the next line when macro encounters an error

If you wish your macro to execute the next line when an error condition occurs, include the statement

On Error Resume Next

in the macro.

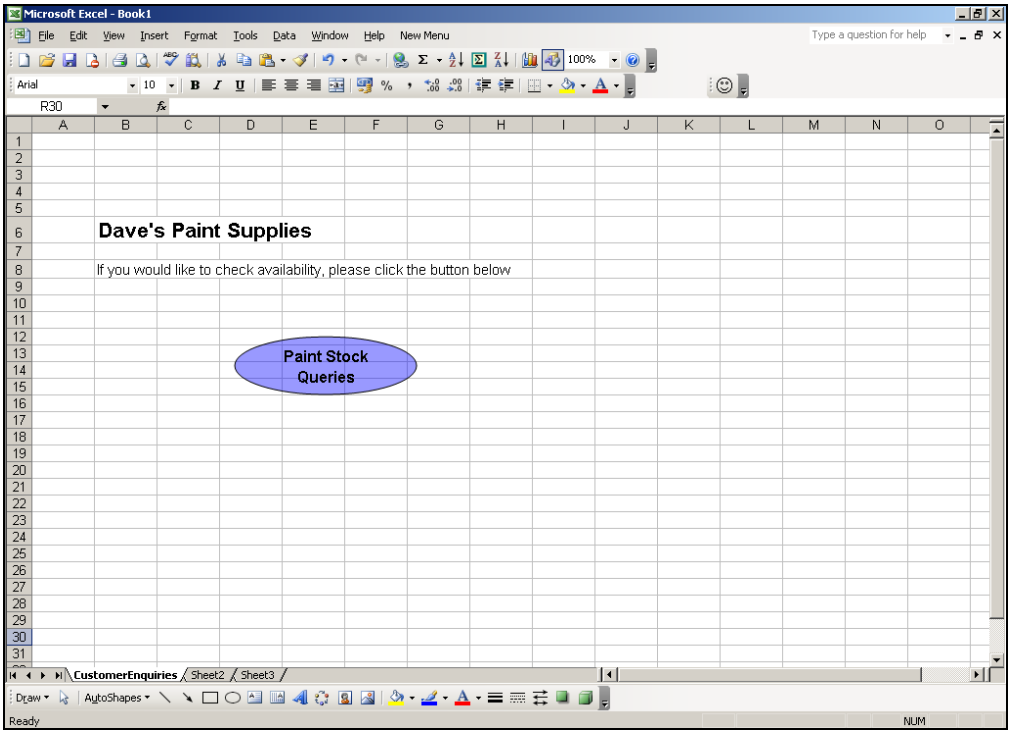
For the macro described above, the effect will be the same as that described in section 13.2.

14 Custom dialog boxes

MsgBox and **InputBox** are two of Excel's built-in dialog boxes. You can create your own dialog boxes for passing information to and from a macro.

A simple example is described below in which a decorator keeps a record of the paint he has in stock and provides a facility for customers to enquire about the availability of a particular colour in either gloss or matt paint.

14.1 The controlling sheet



To create the controlling sheet for this example,

1. Open a new workbook and save it as **Paint.xls**
2. Name the first sheet **CustomerEnquiries** (for example).
3. Make sure that the **Drawing** toolbar is visible (**View** | **Toolbars** | **Drawing**).
4. Click the **Oval** button on the **Drawing** toolbar and draw the oval shape.
5. Right-click on the oval and select **Add Text**.
6. Type

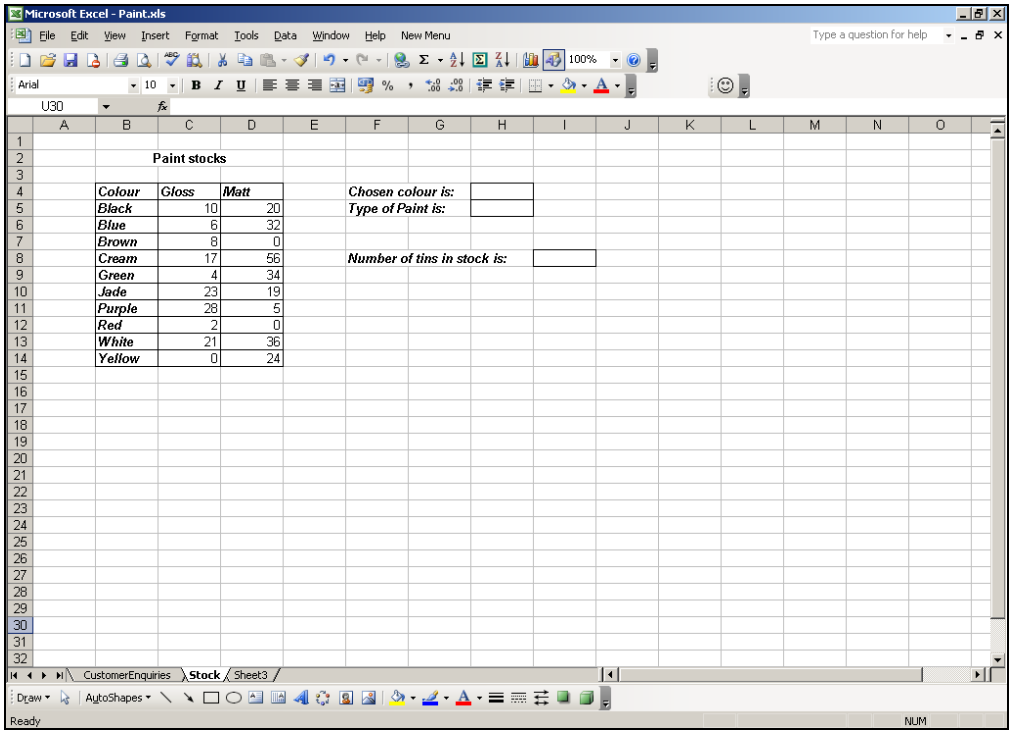
   Paint queries

7. If you would like to give a 3-D effect to the oval, click on the **3-D** button on the **Drawing** toolbar and select one of the settings offered.

14.2 Paint in stock sheet

The record of tins of paint currently in stock is stored on another sheet.

1. Rename **Sheet2** as **Stock**.
2. Enter the data as shown below.



3. Note that the paint colours are sorted alphabetically.
4. In cell **I8**, enter the formula **=VLOOKUP(H4,B5:D14,H5)**

This will give a #N/A result since there is no data in **H4** and **H5** at the moment. Those values will be entered automatically when the customer indicates which paint they are enquiring about.

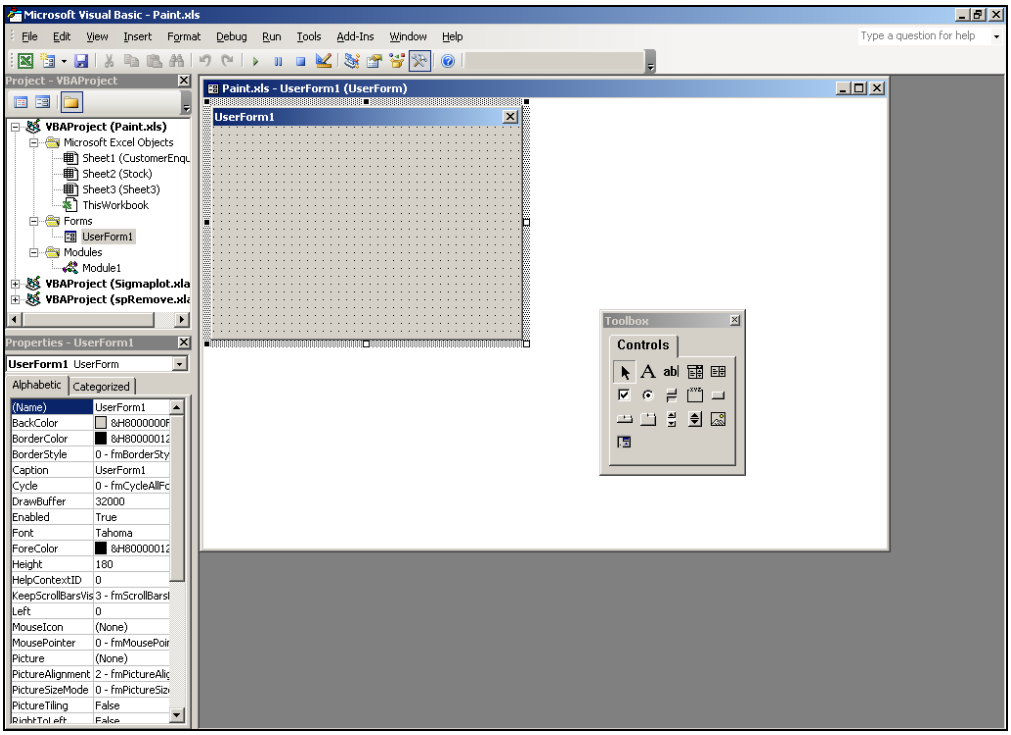
5 If you wish, you can delete any unused sheets.

14.3 Creating the User Form

The user form is the dialog box that is shown to the customer.

1 From the **Tools** menu, select **Macro** and then **Visual Basic Editor**.

2 In the **Microsoft Visual Basic** window, select **UserForm** from the **Insert** menu.



Excel inserts a new user form and displays the **Toolbox**.

3 Find out the names of the various tools in the **Toolbox** by positioning the cursor over them, but ***do not click***.

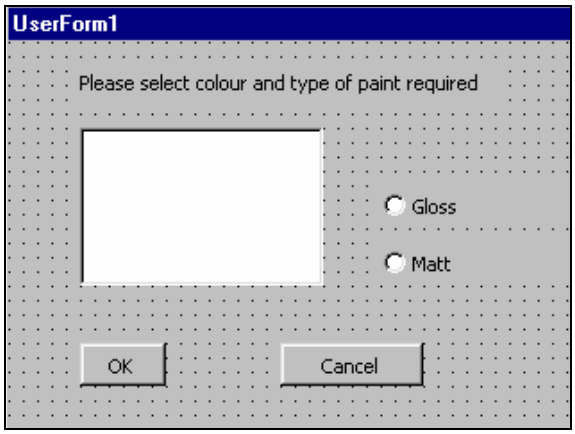
There are two ways of drawing on a form. Either drag a button from the Toolbox to the UserForm window or,

- click a button in the Toolbox window
- position the cursor on the form where the button is to appear

- drag to the size required

The layout of the user form is to be as shown below.

4 Drag the bottom right-hand corner of the **UserForm1** window to make it a little larger.



Instruction towards the top of the form

1 Click on the **Label** control in the **Toolbox**.

2 Draw a wide, shallow rectangle on the form. Its border will be composed of dots.

3 Single left-click (do not double-click) in that rectangle until its outline has diagonal stripes.

4 You will then be able to select the text **Label1** and replace it with

Please select colour and type of paint required

5 Click on a clear region of the form.

OK and Cancel buttons

1 Click on the **CommandButton** control in the **Toolbox**.

2 Drag to create the leftmost button at the bottom of the form.

3 Single left-click (do not double-click) on this button until its outline has diagonal stripes.

4 Select its text and change it to **OK**.

5 Similarly, draw a second button containing the text **Cancel**.

Large rectangle to hold paint colours

1 Click on the **ListBox** control in the **Toolbox**. (If the Toolbox is not visible, click on a blank region of the user form.)

2 Drag to create the white rectangle shown in the example above.

Type of paint

1. Select the **OptionButton** control.
2. Click on the UserForm where the **Gloss** button is to go.
3. Select the text **OptionButton1** and change it to **Gloss**.
4. Repeat all that to create the **Matt** button.

Hint: If you want to precisely line up two items (for example, option buttons), select the first, hold down the Shift key while selecting the second, right-click on one of them and select **Align** from the shortcut menu.

Changing the appearance of text

1. Click on your label (**Please select colour…**).
2. In the **Properties** window (to the left), click on the **Font** entry (**Tahoma**).
3. Click on the … in that box.
4. Select **Tahoma Bold 8pt** and click **OK**.
5. Adjust the size/position of the label if necessary.

Changing Name properties

1. Click on the **Gloss** option button.
2. In the **Properties** window, click on the **Alphabetic** tab and change the (**Name**) property from **OptionButton1** to **ChoseGlossOptionButton**.
3. Click on the **Matt** option button.
4. In the **Properties** window, change the (**Name**) property from **OptionButton2** to **ChoseMattOptionButton**.
5. Click on the **ListBox** (to contain colours eventually).
6. In the **Properties** window, change the (**Name**) property from **ListBox1** to **WhichColourListBox**.
7. Click on the **OK** button.
8. In the **Properties** window, change the (**Name**) property from **CommandButton1** to **OKButton**.
9. Click on the **Cancel** button.
10. In the **Properties** window, change the (**Name**) property from **CommandButton2** to **CancelButton**.

Finally, change the name and caption of **UserForm1** itself:

1. Click in a clear region of **UserForm1** to select it.

2. In the **Properties** window, change the (**Name**) property from **UserForm1** to **PaintDialog**.
3. In the **Properties** window, change the **Caption** setting from **UserForm1** to **WhichPaint?**.

Note: It is not essential to change these default names to more meaningful ones.

14.4 Getting data to and from the list box

The next stage is to arrange for the paint colours to appear in the **ListBox**.

1. On the **UserForm**, make sure that the **ListBox** is selected (click on it if necessary).
2. In the **Properties** pane, select the **Categorized** tab.
3. Scroll down to the **Data** area.
4. Beside the **RowSource** box, type **stock!b5:b14**

Once the customer has selected a colour, that "answer" should be recorded somewhere on the worksheet. In this example, cell **H4** will be used.

5. Beside the **ControlSource** box (two rows above the **RowSource**) type **stock!h4**
6. Click on a blank region of your userform.

The colours should now be listed in the box.

14.5 Setting up the macro

A macro has to be assigned to the **Paint queries** button on the **CustomerEnquiries** worksheet. All it will do is

- set the default value for the **Gloss/Matt** option to **Gloss**
- present the form to the customer

To create that macro,

1. Make sure that the **Visual Basic** window is active.
2. From the **Insert** menu, select **Module**.
3. In that module, enter the macro shown below. The comments will help you to remember what each step does.

```
Sub PaintQueries()
    ' Load the form
     Load PaintDialog
    ' Set up the default value for the Option buttons
    PaintDialog.ChoseGlossOptionButton.Value = True
    ' Set up the default value for the List Box
    PaintDialog.WhichColourListBox.Value = "Black"
    ' Present the dialog box to the customer
    PaintDialog.Show
End Sub
```

This macro has now to be assigned to the button on the **CustomerEnquiries** sheet.

1. Right-click on the edge of the **Paint queries** button on the **CustomerEnquiries** sheet. Be careful to select the oval, not the text box.
2. Select **Assign macro**.
3. Select the **PaintQueries** macro and click **OK**.

14.6 Adding code to the user form

The final stage is to set up procedures to deal with the decisions taken by a customer who uses the form.

1. Display your **PaintDialog** form. (If necessary, in the **Project** pane of the **Microsoft Visual Basic** window, right-click on the **PaintDialog** form and select **View Object**.)

When a customer clicks the **Cancel** button on the dialog box, the form should simply disappear. To achieve that:

2. Right-click on the **Cancel** button.
3. Select **View Code**.
4. Add two lines to the macro as follows

```
Private Sub CancelButton_Click()
    ' Just hide the dialog box without updating the sheet
    PaintDialog.Hide
End Sub
```

When looking up a particular paint in the table of stocks, the second column has to be referred to for **Gloss** and the third for **Matt**. So, **H5** has to have the value **2** if gloss is selected and **3** otherwise.

Once it is known how many tins are in stock, a suitable message can be displayed using **MsgBox**. The calculation of the number of tins is on the **Stock** worksheet in cell **I8**.

This is all controlled from the **OK** button as follows:

5. Return to the userform by selecting **PaintDialog** (**UserForm**) from the **Window** menu.
6. Right-click on the **OK** button.
7. Select **View Code**.
8. Enter the following macro.

```
Private Sub OKButton_Click()
' Use info from dialog box
  If PaintDialog.ChoseGlossOptionButton = True Then
    Range("Stock!H5").Value = 2
  Else
    Range("Stock!H5").Value = 3
  End If
  no_tins = Range("Stock!I8").Value
  If no_tins > 0 Then
    MsgBox "We have " & no_tins & " tins in stock"
  Else
    MsgBox "Sorry, that paint is not in stock at the moment"
  End If
End Sub
```

Note: If the transferring of the colour selected by the customer from the dialog box to the worksheet had not been done in step 5 of section 14.4, it could be done in this macro as described below.

```
thecolour = PaintDialog.WhichColourListBox.Text
Range("Stock!H4").Value = thecolour
```

14.7 Testing

It is important to test that everything is working as planned.

1. Click on the **Paint queries** button on the **CustomerEnquiries** worksheet.
2. Check that the defaults are correct (black and gloss).
3. Select a colour and type of paint and click **OK**.
4. To check that the correct information has been given to you, click on **OK** in the Message Box, close the **WhichPaint?** window and look at the **Stock** sheet.
5. Click on the **Paint queries** button again, try another combination and click **OK**.
6. Try clicking **Cancel**.
7. Click the **Paint queries** button again.
8. Check that the defaults still work.
9. Click **Cancel**.

15 Custom functions

These are also known as **user-defined functions**. They carry out calculations and return an answer. They cannot be recorded, start with **Function** and end with **End Function**.

A very simple example is given below. This function, called **AddThree**, adds three numbers, represented by x, y and z, that will be passed to the function from a macro.

```
Function AddThree(x, y, z)
AddThree = x + y + z
End Function
```

The following procedure uses that function to add 12, 14 and 25, and then displays the answer using MsgBox.

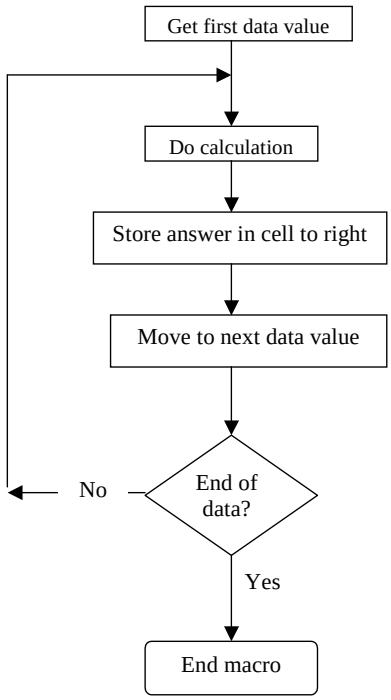
```
Sub Testing()
   a = 12
   b = 14
   c = 25
   answer = AddThree(a, b, c)
   MsgBox "The result is " & answer
End Sub
Function AddThree(x, y, z)
   AddThree = x + y + z
End Function
```

16 Ways of working and some “answers”

16.1 Getting organised

People vary in the way they approach the job of writing a macro. For some, the overall structure of the problem is in their heads and they dive straight in and start typing the actual instructions.

Another approach is to draw a flow diagram, like the one below, before embarking on writing the actual code.



Some people jot a few guidelines on a sheet of paper before starting — perhaps something like

1 Activate the cell containing the first data value

2 Do the calculation

3 Store the answer in the cell to the right of that value

4 Activate the cell containing the next data value

5 Keep repeating (2) to (4) until the end of the data is reached

Different approaches suit different problems and different people but it is very important to approach macro writing in an organised fashion.

16.2 Solutions to exercises

The macros listed in this section are *suggestions* as to how you might solve particular problems. There is not just *one* correct way of writing a macro; there are usually several possible approaches to any situation. So, if your macro works correctly and is efficient, it does not matter if it looks different from the corresponding one listed here.

These solutions assume that the correct workbook and worksheet will be active when each macro is run. If this is not the case then an instruction such as **Workbooks("MacroData.xls").Worksheets("Names").Activate** or **Workbooks("MacroData.xls").Sheets("Names").Activate** should also be included.

```
Sub Address_abs()                                    ' Section 2.1
' Address_abs Macro
' Enters address starting in cell B3
    Range("B3").Select
    ActiveCell.FormulaR1C1 = "D.Morgan"
    Range("B4").Select
    ActiveCell.FormulaR1C1 = "IT Service"
    Range("B5").Select
    ActiveCell.FormulaR1C1 = "University of Durham"
    Range("B3:B5").Select
    Selection.Font.Bold = True
    Selection.Font.Italic = True
    Range("B6").Select
End Sub
```

```
Sub Address_rel()                                    ' Section 2.3
' Address_rel Macro
' Enters address at the active cell position
' Keyboard Shortcut: Ctrl+Shift+A
    ActiveCell.FormulaR1C1 = "D.Morgan"
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "IT Service"
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "University of Durham"
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "DH1 3LE"
    ActiveCell.Offset(-3, 0).Range("A1:A4").Select
    Selection.Font.Bold = True
    ActiveCell.Offset(4, 0).Range("A1").Select
End Sub
```

```
Sub MyFirst()                                          ‘ Section 7.1
   Sheets("Text").Select
   Range("B2").Select
   ActiveCell.FormulaR1C1 = "I can write macros!"
   Selection.Font.Bold = True
End Sub
```

```
Sub DeleteValues()                                     ‘ Section 8.8
' Exercise 1
   Range("C5:E5").ClearContents
End Sub
```

```
Sub EnterValue()                                       ‘ Section 8.8
' Exercise 2
   Cells(7, 4).Value = "Year 2000"
End Sub
```

```
Sub DeleteRow()                                        ‘ Section 8.8
' Exercise 3
   Rows(6).Delete
End Sub
```

```
Sub PassFail()                                         ‘ Section 9.1
' Exercise on using If Then Else
   If Cells(5, 6) > 49 Then
      Cells(5, 7) = "Pass"
   Else
      Cells(5, 7) = "Fail"
   End If
End Sub
```

```
Sub SSAW1()                                            ‘ Section 9.2
'Exercise on using Select Case
’Another version shown below
   n = Range("R3").Value
   Select Case n
   Case 3 To 5
      Range("S3").Value = "Spring"
   Case 6 To 8
      Range("S3").Value = "Summer"
   Case 9 To 11
      Range("S3").Value = "Autumn"
   Case 12, 1, 2
      Range("S3").Value = "Winter"
   End Select
End Sub
```

```
Sub SSAW2()                                          ‘ Section 9.2
'Exercise on using Select Case
   Select Case Range("R3")
   Case 3 To 5
      Range("S3") = "Spring"
   Case 6 To 8
      Range("S3") = "Summer"
   Case 9 To 11
      Range("S3") = "Autumn"
   Case 12, 1, 2
      Range("S3") = "Winter"
   End Select
End Sub
```

```
Sub Leaflet1()                                       ‘ Section 9.3.1
’Another version shown below
'Exercise on using If with And
   If Cells(3, 23) > 20 And Cells(3, 23) < 36 Then
      Cells(3, 24) = "Yes"
   Else
      Cells(3, 24) = "No"
   End If
End Sub
```

```
Sub Leaflet2()                                       ‘ Section 9.3.1
'Exercise on using If with And
   age = Range("W3")
   If age >= 21 And age <= 35 Then
      Range("X3") = "Yes"
   Else
      Range("X3") = "No"
   End If
End Sub
```

```
Sub AddNumbers1()                                    ‘ Section 11.2
'Exercise on using Do While loop
'Do Until version listed below
   total = 0
   r = 3
   Do While Cells(r, 7) <> "End of data"
      total = total + Cells(r, 7)
      r = r + 1
   Loop
   MsgBox "The total is " & total
End Sub
```

```
Sub AddNumbers2()                                    ‘ Section 11.2
'Exercise on using Do Until loop
   total = 0
   r = 3
   Do
      total = total + Cells(r, 7)
      r = r + 1
   Loop Until Cells(r, 7) = "End of data"
   MsgBox "The total is " & total
End Sub
```

```
Sub Names()                                          ' Section 11.3
'Exercise on using For Next loop
   Cells(20, 4) = "Full Name"
   Cells(20, 4).Font.Bold = True
   For r = 21 To 26
      Cells(r, 4) = Cells(r, 3) & " " & Cells(r, 2)
   Next r
End Sub
```

```
Sub LongList1()                                      ' Section 12
'Exercise on using list of unspecified length - cells not selected
'Two other versions shown below
   n = Cells(2, 1).End(xlDown).Row
   For r = 3 To n
     Cells(r, 3) = Cells(r, 2) & " " & Cells(r, 1)
   Next r
End Sub
```

```
Sub LongList2()                                      ' Section 12
'Exercise on using list of unspecified length - cells not selected
'Another version shown below
   r = 3
   Do While Cells(r, 1) <> ""
      Cells(r, 3) = Cells(r, 2) & " " & Cells(r, 1)
      r = r + 1
   Loop
End Sub
```

```
Sub LongList3()                                      ' Section 12
'Exercise on using list of unspecified length - cells selected
   Range("A3").Select
   Do While ActiveCell <> ""
      ActiveCell.Offset(0, 2) = ActiveCell.Offset(0, 1) & _
        " " & ActiveCell.Value
      ActiveCell.Offset(1, 0).Select
   Loop
End Sub
```