**Working with macros**

**Recording a macro: a simple example**

This example demonstrates how to record a simple macro that inserts your name into the active cell. To create the macro, start with a new workbook and follow these steps:

1. Select an empty cell.
2. Choose Developer ➪ Code ➪ Record Macro. The Record Macro dialog box appears.
3. Enter a new single-word name for the macro to replace the default Macro1 name. For example, type MyName as the name.
4. Assign this macro to the shortcut key Ctrl+Shift+N by entering an uppercase N in the Shortcut Key field.
5. Make sure that This Workbook is selected in the Store Macro In field.
6. Click OK to close the Record Macro dialog box and begin recording your actions.
7. Type your name into the selected cell and then press Enter.
8. Choose Developer ➪ Code ➪ Stop Recording (or click the Stop Recording button on the status bar).

**Examining the macro**

The macro was recorded in a new module named Module1. To view the code in this module,

you must activate the VBE. You can activate the VBE in either of two ways:

* Press Alt+F11.
* Choose Developer ➪ Code ➪ Visual Basic.

In the VBE, the Project window displays a list of all open workbooks and add-ins. This list is displayed as a tree diagram, which you can expand or collapse. The code you recorded previously is stored in Module1 in the Modules folder of the current workbook. When you double-click Module1, the code in the module appears in the Code window.

The macro should look something like this (with your name substituted in, of course):



The macro recorded is a Sub procedure that is named MyName. The statements tell Excel what to do when the macro is executed. Notice that Excel inserted some comments at the top of the procedure. These comments are based on information that appeared in the Record Macro dialog box. These comment lines (which begin with an apostrophe) aren’t really necessary, and deleting them has no effect on how the macro runs. If you ignore the comments, you’ll see that this procedure has only two VBA statements:

ActiveCell.FormulaR1C1 = "Name Surname"

Range("A2").Select

The first statement causes the name that you typed while recording the macro to be inserted into the active cell. The FormulaR1C1 part is a property of the Range object. When you press Enter in a cell, Excel moves down one cell (unless you’ve change the default behaviour). You can guess from this code that the active cell was A2 when the macro was recorded.

**Editing the macro**

After you record a macro, you can make changes to it (although you must know what you’re doing). For example, assume that you don’t want to select A2, but rather you want to select the cell below the active cell. Press Alt+F11 to activate the VBE window. Then activate Module1 and change the second statement to the following:

ActiveCell.Offset(1, 0).Select

The edited macro appears as follows:



**Controlling execution**

VBA uses many constructs that are found in most other programming languages. These constructs are used to control the flow of execution. This section introduces a few of the more common programming constructs.

***The If-Then construct***

One of the most important control structures in VBA is the If-Then construct, which gives your applications decision-making capability. The basic syntax of the If-Then structure is as follows:

If condition Then statements [Else elsestatements]

In plain English, if a condition is true, then a group of statements will be executed. If you include the Else clause, then another group of statements will be executed if the condition is not true.

The following is an example (which doesn’t use the optional Else clause). This procedure checks the active cell. If it contains a negative value, the cell’s font color is changed to red. Otherwise, nothing happens:

Sub CheckCell()

If ActiveCell.Value < 0 Then ActiveCell.Font.Color = vbRed

End Sub

Here’s another multiline version of that procedure that uses an Else clause. Because it uses multiple lines, you must include an End If statement. This procedure colors the active cell text red if it’s a negative value and green otherwise:

Sub CheckCell()

If ActiveCell.Value < 0 Then

ActiveCell.Font.Color = vbRed

Else

ActiveCell.Font.Color = vbGreen

End If

End Sub

***For-Next loops***

You can use a For-Next loop to execute one or more statements a number of times. Here’s an example of a For-Next loop:

Sub SumSquared()

Total = 0

For Num = 1 To 10

Total = Total + (Num ^ 2)

Next Num

MsgBox Total

End Sub

This example has one statement between the For statement and the Next statement. This single statement is executed ten times. The variable Num takes on successive values of 1, 2, 3, and so on, up to 10. The variable Total stores the sum of Num squared added to the previous value of Total. The result is a value that represents the sum of the first ten integers squared. This result is displayed in a message box.

***Do loops***

For-Next loops execute a set of statements a particular number of times. Do loops execute a set of statements until a particular condition exists or stops existing.

Sub SumSquaredTo500()

Total = 0

num = 0

Do

num = num + 1

Total = Total + (num ^ 2)

Loop Until Total >= 500

MsgBox num & Space(1) & Total

End Sub

This procedure keeps summing squares until the total gets above 500. With Do loops, you can check the condition on the Do line or the Loop line, but not both. The four options are as follows:

* Do Until
* Do While
* Loop Until
* Loop While

***The With-End With construct***

A construct that you sometimes encounter if you record macros is the With-End With construct. This is a shortcut way of dealing with several properties or methods of the same object. The following is an example:

Sub AlignCells()

With Selection

.HorizontalAlignment = xlCenter

.VerticalAlignment = xlCenter

.WrapText = False

.Orientation = xlHorizontal

End With

End Sub

The following macro performs the same operations but doesn’t use the With-End With construct:

Sub AlignCells()

Selection.HorizontalAlignment = xlCenter

Selection.VerticalAlignment = xlCenter

Selection.WrapText = False

Selection.Orientation = xlHorizontal

End Sub

***The Select Case construct***

The Select Case construct is useful for choosing among two or more options. The following example demonstrates the use of a Select Case construct. In this example, the active cell is checked. If its value is less than 0, it’s colored red. If it’s equal to 0, it’s colored blue. If the value is greater than 0, it’s colored black:

Sub CheckCell()

Select Case ActiveCell.Value

Case Is < 0

ActiveCell.Font.Color = vbRed

Case 0

ActiveCell.Font.Color = vbBlue

Case Is > 0

ActiveCell.Font.Color = vbBlack

End Select

End Sub

Any number of statements can go below each Case statement, and they all are executed if the case is true.

**A macro that can’t be recorded**

The following is a VBA macro that can’t be recorded because it uses programming concepts that must be entered manually. This macro creates a list of all formulas on the active sheet. The list is stored on a new worksheet:

Sub ListFormulas()

' Create a range variable

Set InputRange = ActiveSheet.UsedRange

' Add a new sheet and save in a variable

Set OutputSheet = Worksheets.Add

' Variable for the output row

OutputRow = 1

' Loop through the range

For Each cell In InputRange

If cell.HasFormula Then

OutputSheet.Cells(OutputRow, 1) = "'" & cell.Address

OutputSheet.Cells(OutputRow, 2) = "'" & cell.Formula

OutputRow = OutputRow + 1

End If

Next Cell

End Sub

Although this macro may look complicated, it’s fairly simple when you break it down. Here’s how it works:

1. The macro creates an object variable named InputRange. This variable corresponds to the used range on the active sheet (avoiding the need to check every cell).
2. It then adds a new worksheet and assigns the worksheet to an object variable named OutputSheet. The OutputRow variable is set to 1. This variable is incremented later.
3. The For-Next loop examines each cell in the InputRange. If the cell has a formula, the cell’s address and formula are written to the OutputSheet. The OutputRow variable is also incremented.

As macros go, this example is okay, but it’s certainly not perfect. It’s not very flexible, and it doesn’t include error handling. For example, if the workbook structure is protected, trying to add a new sheet will cause an error.

**Using UserForm Controls in a Worksheet**

The main reason to use controls on a worksheet is to make it easier for the user to provide input. For example, if you create a model that uses one or more input cells, you can create controls to allow the user to select values for the input cells.

Adding controls to a worksheet requires much less effort than creating a dialog box. In addition, you may not have to create any macros because you can link a control to a worksheet cell. For example, if you insert a CheckBox control on a worksheet, you can link it to a particular

cell. When the CheckBox is checked, the linked cell displays TRUE. When the CheckBox is not checked, the linked cell displays FALSE.

Figure shows an example that uses three types of controls: a Checkbox, two sets of OptionButtons, and a ScrollBar. The user’s selections are used to display a loan amortization schedule on another worksheet. The workbook is interactive, but it uses no macros.



Adding controls to a worksheet can be a bit confusing because Excel offers two different sets of controls, both of which you access by choosing Developer ➪ Controls ➪ Insert:

* Form controls These controls are unique to Excel.
* ActiveX controls These controls are a subset of those that are available for use on UserForms.

Figure shows the controls that appear when you choose Developer ➪ Controls ➪ Insert. When you move your mouse pointer over a control, Excel displays a ScreenTip that identifies the control.



To add to the confusion, many controls are available from both sources. For example, a control named ListBox is listed in both Forms controls and ActiveX controls. However, they’re two entirely different controls. In general, Forms controls are easier to use, but ActiveX controls provide more flexibility. ActiveX Controls:





Next part will be about ActiveX controls only.

**Adding a control**

To add a control to a worksheet, choose Developer ➪ Controls ➪ Insert. From the Insert drop-down list, click the control you want to use and then drag it into the worksheet to create the control. You don’t need to be too concerned about the exact size or position of the control because you can modify those properties at any time.

***Learning about Design mode***

When you add a control to a worksheet, Excel goes into Design mode. In this mode, you can adjust the properties of any controls on your worksheet, add or edit macros for the control, or change the control’s size or position. When Excel is in Design mode, the controls aren’t enabled. To test the controls, you must exit Design mode by clicking the Design Mode button. When you’re working with controls, you’ll probably need to switch in and out of Design mode frequently.

***Adjusting properties***

Every control that you add has various properties that determine how it looks and behaves. You can adjust these properties only when Excel is in Design mode. When you add a control to a worksheet, Excel enters Design mode automatically. If you need to change a control after you exit Design mode, click the Design Mode button in the Controls section of the Developer tab. To change the properties for a control, follow these steps:

1. Make sure Excel is in Design mode.
2. Click the control to select it.
3. If the Properties window isn’t visible, click the Properties icon in the Controls section of the Developer tab. The Properties window appears.
4. Select the property and make the change.



The manner in which you change a property depends on the property. Some properties display a drop-down list from which you can select from a list of options. Others (such as Font) provide a button that displays a dialog box when clicked. Other properties require you to type the property value. When you change a property, the change takes effect immediately. To find out about a particular property, select the property in the Properties window and press F1.

The Properties window has two tabs. The Alphabetic tab displays the properties in alphabetical order. The Categorized tab displays the properties by category. Both tabs show the same properties; only the order is different.

***Using common properties***

Each control has its own unique set of properties. However, many controls share properties. This section describes some properties that are common to all or many controls. Properties shared by multiple controls:





***Linking controls to cells***

Often, you can use ActiveX controls in a worksheet without using macros. Many controls have a LinkedCell property, which specifies a worksheet cell that is linked to the control. For example, you may add a SpinButton control and specify cell B1 as its LinkedCell property. After doing so, cell B1 contains the value of the SpinButton, and clicking the SpinButton changes the value in cell B1. You can, of course, use the value contained in the linked cell in your formulas.

***Creating macros for controls***

To create a macro for a control, you must use the Visual Basic Editor (VBE). The macros are stored in the code module for the sheet that contains the control. For example, if you place an ActiveX control on Sheet2, the VBA code for that control is stored in the Sheet2 code module. Each control can have a macro to handle any of its events. For example, a CommandButton control can have a macro for its Click event, its DblClick event, and various other events.



The control’s name appears in the upper-left portion of the code window, and the event appears in the upper-right area. If you want to create a macro that executes when a different event occurs, select the event from the list in the upper-right area. The following steps demonstrate how to insert a CommandButton and to create a simple macro that displays a message when the button is clicked:

1. Choose Developer ➪ Controls ➪ Insert.
2. Click the CommandButton tool in the ActiveX Controls section.
3. Click and drag in the worksheet to create the button. Excel automatically enters Design mode.
4. Double-click the button. The VBE is activated, and an empty procedure is created for the button’s Click event.
5. Enter the following VBA statement before the End Sub statement: MsgBox "Hello, it's " & Time
6. Press Alt+F11 to return to Excel.
7. (Optional) Adjust any other properties for the CommandButton using the Properties window. Choose Developer ➪ Controls ➪ Properties if the Properties window isn’t visible.
8. Click the Design Mode button in the Developer ➪ Controls section to exit design mode.

After you perform the preceding steps, when you click the CommandButton, the message box appears and displays the current time.

**Reviewing the Available ActiveX Controls**

The following sections describe the ActiveX controls that are available for use in your worksheets.

***CheckBox***

A CheckBox control is useful for getting a binary choice: yes or no, true or false, on or off, and so on. The following is a description of the most useful properties of a CheckBox control:

* Accelerator A letter that enables the user to change the value of the control by using the keyboard. For example, if the accelerator is A, pressing Alt+A changes the value of the CheckBox control. The accelerator letter is underlined in the caption of the control.
* LinkedCell The worksheet cell that’s linked to the CheckBox. The cell displays TRUE if the control is checked or FALSE if the control is not checked.

***ComboBox***

A ComboBox control is a combination of a TextBox and a ListBox. It acts as a TextBox because the user can type in it just like a TextBox, even if what they type isn’t on the list. It acts as a ListBox because when you click its drop-down arrow, a list of available items appears. The following is a description of the most useful properties of a ComboBox control:

* BoundColumn If the ListFillRange contains multiple columns, this property determines which column contains the returned value.
* ColumnCount This specifies the number of columns to display in the list.
* LinkedCell This specifies the worksheet cell that displays the selected item.
* ListFillRange This specifies the worksheet range that contains the list items.
* ListRows This specifies the number of items to display when the list drops down.
* ListStyle This determines the appearance of the list items.
* Style This determines whether the control acts like a drop-down list or a ComboBox. A drop-down list doesn’t allow the user to enter a new value.

You can also create a drop-down list directly in a cell by using data validation.

***CommandButton***

A CommandButton control is used to execute a macro. When a CommandButton is clicked, it executes an event procedure with a name that consists of the CommandButton name, an underscore, and the word Click. For example, if a CommandButton is named MyButton, clicking it executes the macro named MyButton\_Click. This macro is stored in the code module for the sheet that contains the CommandButton.

***Image***

An Image control is used to display an image. These are the most useful properties of an Image control:

* AutoSize If TRUE, the Image control is resized automatically to fit the image.
* Picture This is the path to the image file. Click the button in the Properties window, and Excel displays a dialog box so that you can locate the image. Or, copy the image to the Clipboard, select the Picture property in the Properties window, and press Ctrl+V.
* PictureSizeMode This determines how the picture is changed when the container size is different than the picture.

***Label***

A Label control simply displays text. Like on a UserForm, it’s used to describe other controls. You can also use its Click event to activate other controls with an accelerator key.

***ListBox***

A ListBox control presents a list of items, and the user can select an item (or multiple items). It’s similar to a ComboBox. The main difference is that a ListBox doesn’t require you to click a drop-down arrow to display more than one item at a time. The following is a description of the most useful properties of a ListBox control:

* BoundColumn If the list contains multiple columns, this property determines which column contains the returned value.
* ColumnCount This specifies the number of columns to display in the list.
* IntegralHeight This is TRUE if the height of the ListBox adjusts automatically to display full lines of text when the list is scrolled vertically. If FALSE, the ListBox may display partial lines of text when it’s scrolled vertically.
* LinkedCell This specifies the worksheet cell that displays the selected item.
* ListFillRange This specifies the worksheet range that contains the list items.
* ListStyle This determines the appearance of the list items.
* MultiSelect This determines whether the user can select multiple items from the list.

***OptionButton***

OptionButton controls are useful when the user needs to select from a small number of items. OptionButton controls are always used in groups of at least two. The following are the most useful properties of an OptionButton control:

* Accelerator A letter that lets the user select the option by using the keyboard. For example, if the accelerator for an OptionButton is C, pressing Alt+C selects the control.
* GroupName A name that identifies an OptionButton as being associated with other OptionButtons with the same GroupName property.
* LinkedCell The worksheet cell that’s linked to the OptionButton. The cell displays TRUE if the control is selected or FALSE if the control isn’t selected.

***ScrollBar***

A ScrollBar control is useful for specifying a cell value. The following are the most useful properties of a ScrollBar control:

* Value The current value of the control
* Min The minimum value for the control
* Max The maximum value for the control
* LinkedCell The worksheet cell that displays the value of the control
* SmallChange The amount that the control’s value is changed by clicking the arrows
* LargeChange The amount that the control’s value is changed by clicking in the scroll area

The ScrollBar control is most useful for selecting a value that extends across a wide range of possible values.

***SpinButton***

A SpinButton control lets the user select a value by clicking the control, which has two arrows (one to increase the value and the other to decrease the value). A SpinButton can display either horizontally or vertically. The following are the most useful properties of a SpinButton control:

* Value The current value of the control.
* Min The minimum value of the control.
* Max The maximum value of the control.
* LinkedCell The worksheet cell that displays the value of the control.
* SmallChange The amount that the control’s value is changed by a click. Usually, this property is set to 1, but you can make it any value.

***TextBox***

On the surface, a TextBox control may not seem useful. After all, it simply contains text— you can usually use worksheet cells to get text input. In fact, TextBox controls are useful not so much for input control as for output control. Because a TextBox can have scrollbars, you can use a TextBox to display a great deal of information in a small area. The following are the most useful properties of a TextBox control:

* AutoSize This determines whether the control adjusts its size automatically, based on the amount of text.
* IntegralHeight If TRUE, the height of the TextBox adjusts automatically to display full lines of text when the list is scrolled vertically. If FALSE, the ListBox may display partial lines of text when it’s scrolled vertically.
* MaxLength This determines the maximum number of characters allowed in the TextBox. If 0, no limit exists on the number of characters.
* MultiLine If TRUE, the TextBox can display more than one line of text.
* TextAlign This determines the way the text is aligned in the TextBox.
* WordWrap This determines whether the control allows word wrap.
* ScrollBars This determines the type of ScrollBars for the control: horizontal, vertical, both, or none.

***ToggleButton***

A ToggleButton control has two states: on and off. Clicking the button toggles between these two states, and the button changes its appearance to indicate its current state. Its value is either TRUE (pressed) or FALSE (not pressed). You can often use a ToggleButton in place of a CheckBox control.