

# UserForm Code Reference

## Basic Coding

This document explains the basic requirements for coding UserForms and illustrates some of the more common techniques.

### Opening and Closing a UserForm

#### *Opening a UserForm*

The code for opening a UserForm does not reside in the UserForm's own code module. Create a separate module and insert the following macro:

```
Sub OpenMyForm()  
    UserForm1.Show  
End Sub
```

Where *UserForm1* is the name of the UserForm (check its *Name* property in the Properties Window). The name of the macro is not important. The macro can then be run in the normal way from the Tools menu or from a custom button or menu item.

If you want a UserForm to open automatically when its host document opens place the code in the document's *Open* event procedure. For example, in a Microsoft Excel workbook this should be in the *Workbook\_Open* event in the *ThisWorkbook* module.

```
Private Sub Workbook_Open()  
    UserForm1.Show  
End Sub
```

In Microsoft Word use the *Document\_Open* event in the *ThisDocument* module:

```
Private Sub Document_Open()  
    UserForm1.Show  
End Sub
```

There is a *Load* method which loads the UserForm into the computer's memory but does not make it visible. Since the *Show* method automatically loads the form anyway if it is not already loaded, the *Load* method is seldom used.

#### *Closing a UserForm*

There are two ways to close a UserForm. You can *Unload* it or *Hide* it. Such procedures are usually run from a button on the form itself.

When using the *Unload* method from in the form itself you can refer to the form using the keyword *Me*. If you are referring to a different form you must refer to it by its name:

```
Unload Me    or    Unload UserForm1
```

The *Unload* statement closes the form, removes it from the computer's memory and returns control of the host file to the user. When the form is next opened it will have lost any data that had been entered into it.

Using the *Hide* statement removes a UserForm from view, returning control of the host file to the user, but the form does not actually close, being retained in the computer's memory until it is needed again. Use this method if you want to temporarily hide the form so that when it is opened again it appears exactly as it was when it was hidden with textboxes filled, combobox choices retained etc. It is written as follows:

```
Me.Hide    or    UserForm1.Hide
```

### Preparing the Form

#### *Setting the Value of a Control*

Sometimes it is necessary to prepare a form before it is shown to the user. This usually involves setting the start values of various controls and filling the lists of comboboxes and listboxes. This code must be placed in the form's *UserForm\_Initialize* procedure.

For example, you might want to automatically enter today's date into a textbox. This example uses the VBA *Date* function to do that:

```
Me.TextBox1.Value = Date
```

Or, you might read existing values from cells on an Excel worksheet:

```
Me.TextBox1.Value = Worksheets("Sheet1").Range("A1").value
```

### *Building a ComboBox or TextBox List*

If you are working in Microsoft Excel, the most convenient way to create the list in a combobox or list box is to set the control's *RowSource* property to a named range in the workbook. But it is not always convenient to do this and, if you are working in another program such as Microsoft Word you have to find another way anyway. The solution is to build the list with code in the *UserForm\_Initialize* procedure.

```
Private Sub UserForm_Initialize()
    With Me.ComboBox1
        .AddItem "Monday"
        .AddItem "Tuesday"
        .AddItem "Wednesday"
        .AddItem "Thursday"
        .AddItem "Friday"
        .AddItem "Saturday"
        .AddItem "Sunday"
    End With
End Sub
```

When there are many items to add this, and there is some sort of numerical relationship between them, it is often possible to use a loop to do the job. This example creates a list of numbers from 100 to 1000 in steps of 50 (e.g. 50, 100, 150, 200 etc.)

```
Private Sub UserForm_Initialize()
    Dim i As Integer
    For i = 50 To 1000 Step 50
        Me.ComboBox1.AddItem i
    Next i
End Sub
```

This example creates a list of year numbers 25 years either side of the current year.

```
Private Sub UserForm_Initialize()
    Dim i As Integer
    For i = -25 To 25
        Me.ComboBox1.AddItem Year(Date) + i
    Next i
End Sub
```

With a little imagination a wide range of items can be listed this way.

## Writing Data from the UserForm to an Excel Workbook

### *Writing to a Range*

The most common practice is to write data from a control into a specific cell on a worksheet. This is done by setting the value of the cell to the value of the control.

```
Range("A1").Value = Me.TextBox1.Value
```

If you do not specify a workbook name the active workbook is assumed. Similarly if you do not specify the name of the worksheet, the active sheet is assumed. It is usually good practice to specify at least the sheet name, for example:

```
Worksheets("Sheet1").Range("A1").Value = Me.TextBox1.Value    or
```

```
Range("Sheet1!A1").Value = Me.TextBox1.Value
```

Alternatively you can write to a named range in which case it is not necessary to specify the worksheet since this is implicit in the named range definition:

```
Range("MyRangeName").Value = Me.TextBox1.Value
```

In some cases it is sufficient to simply write to the currently selected cell:

```
ActiveCell.Value = Me.TextBox1.Value
```

The *Offset* property of a range is very useful when writing the values from multiple controls on the UserForm into a row or column of cells. Use a *With* statement to avoid having to repeat code and increment the second (column) argument of the *Offset* property:

```
With Range("A1")
    .Offset(0, 0).Value = Me.TextBox1.Value
    .Offset(0, 1).Value = Me.ComboBox1.Value
    .Offset(0, 2).Value = Me.CheckBox1.Value
End With
```

Writing down a column is almost the same but this time increment the first (row) argument of the *Offset* property:

```
With Range("A1")
    .Offset(0, 0).Value = Me.TextBox1.Value
    .Offset(1, 0).Value = Me.ComboBox1.Value
    .Offset(2, 0).Value = Me.CheckBox1.Value
End With
```

### Using Loops to Write Multiple Values

When there are multiple values to write to a worksheet you can often streamline your code by using a loop. Suppose you have ten textboxes. Give them all similar names, differing only by an incrementing number at the end of the name. You can then make use of this number to write to a row (or column) of cells using a *For...Next* loop:

```
Dim i As Integer
For i = 1 To 10
    Worksheets("Sheet1").Range("A1").Offset(0, i - 1).Value = _
        Me.Controls("TextBox" & i).Value
Next i
```

In this loop as the variable "i" is incremented for each circuit of the loop, it identifies each textbox in turn and is used to specify a different column in the same row for each piece of data. The example uses "i - 1" to make the first column have an offset of zero, and so on.

This example could, of course, be used with any kind of control, not just textboxes. For a mixed selection of control types, you could give them a common name such as *Control1*, *Control2* etc.

### Finding the Next Row on the Worksheet

To write on to a worksheet that already contains rows of data you need to first ascertain how many rows already exist. Use the *CurrentRegion* property of the start cell (the first cell in the column you are writing to):

```
Dim RowCount As Long
RowCount = Worksheets("Sheet1").Range("A1").CurrentRegion.Rows.Count
```

Having stored this value in a variable, it can be used in conjunction with the row argument of the offset property to insert data in the next available row. The same technique could be applied to columns. Here is an example of writing to the next row:

```
Dim RowCount As Long
RowCount = Worksheets("Sheet1").Range("A1").CurrentRegion.Rows.Count
With Range("A1")
    .Offset(RowCount, 0).Value = Me.TextBox1.Value
    .Offset(RowCount, 1).Value = Me.ComboBox1.Value
    .Offset(RowCount, 2).Value = Me.CheckBox1.Value
End With
```

## Writing to a Word Document

When writing to a Word document you can choose to write to the current insertion point (i.e. the position of the user's cursor in the document) or to one or more bookmarks. When writing to the insertion point use the *TypeText* method to enter the value of the UserForm control:

```
Selection.TypeText Text:=Me.TextBox1
```

You can enter several items at a time by entering paragraph breaks:

```
With Selection
    .TypeText Me.TextBox1.Value
    .TypeParagraph
    .TypeText Me.ComboBox1.Value
    .TypeParagraph
    .TypeText Me.CheckBox1
End With
```

To enter line breaks add the ASCII code for a line break:

```
With Selection
    .TypeText Me.TextBox1.Value & Chr(11)
    .TypeText Me.ComboBox1.Value & Chr(11)
    .TypeText Me.CheckBox1
End With
```

Since the layout of Word documents allow great flexibility there are many different ways to enter data. This example writes the UserForm's data to the document with the items separated by a comma and space:

```
Selection.TypeText Me.TextBox1.Value & ", " & _
    Me.ComboBox1.Value & ", " & Me.CheckBox1.Value
```

Each of the preceding examples enters text at the insertion point. For greater control, place bookmarks in your document where you want the various items to be entered. To enter a single item into a bookmark:

```
ActiveDocument.Bookmarks("Bookmark1").Range.Text = Me.TextBox1.Value
```

A *With* statement is useful when entering values into several bookmarks:

```
With ActiveDocument
    Bookmarks("Bookmark1").Range.Text = Me.TextBox1.Value
    Bookmarks("Bookmark2").Range.Text = Me.ComboBox1.Value
    Bookmarks("Bookmark3").Range.Text = Me.CheckBox1.Value
End With
```