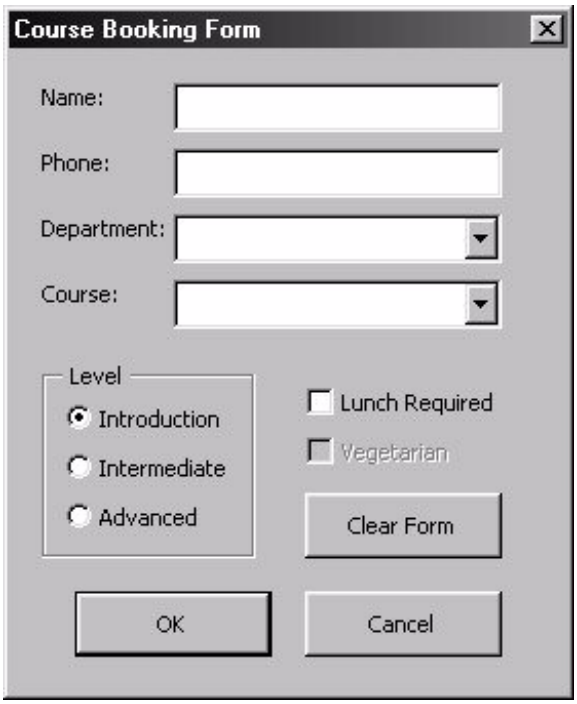
Excel VBA Course Notes: 4. User Forms (File: VBA04-UserForms.xls)

The Course Booking Form



The **Course Booking Form** is a simple form illustrating the principles of UserForm design and the associated VBA coding.

It uses a selection of *controls* including text boxes, combo boxes, option buttons grouped in a frame, check boxes and command buttons.

When the user clicks the OK button their input is entered into the next available row on the worksheet.

Description of the Form:

There are two simple text boxes (*Name:* and *Phone:*) into which the user can type free text, and two combo boxes (*Department* and *Course*) that let the user to pick an item from the list.

There are three option buttons (*Introduction*, *Intermediate* and *Advanced*) grouped in a frame (*Level*) so that the user can choose only one of the options.

There are two check boxes (*Lunch Required* and *Vegetarian*) that, because they are not grouped in a frame, can both be chosen if required. However, if the person making the booking does not want lunch we do not nee d to know whether or not they are vegetarian. So, the *Vegetarian* check box is greyed-out until required.

There are three command buttons (*OK*, *Cancel* and *Clear Form*) each of which performs a pre-defined function when clicked.

The Control Properties Settings:

Control	***Type***	***Property***	***Setting***
UserForm	UserForm	*Name*	frmCourseBooking
		Caption	Course Booking Form
Name	Text Box	*Name*	txtName
Phone	Text Box	*Name*	txtPhone
Department	Combo Box	*Name*	cboDepartment
Course	Combo Box	*Name*	cboCourse
Level	Frame	*Name*	fraLevel
		Caption	Level
Introduction	Option Button	*Name*	optIntroduction
Intermediate	Option Button	*Name*	optIntermediate
Advanced	Option Button	*Name*	optAdvanced
Lunch Required	Check Box	*Name*	chkLunch
Vegetarian	Check Box	*Name*	chkVegetarian
		Enabled	False
OK	Command Button	*Name*	cmdOk
		Caption	OK
		Default	True
Cancel	Command Button	*Name*	cmdCancel
		Caption	Cancel
		Cancel	True
Clear Form	Command Button	*Name*	cmdClearForm

Building the Form

If you want to build the form yourself, simply copy the layout shown in the illustration above. Follow the steps below:

1. Open the workbook that you want the form to belong in (UserForms like macros have to be attached to a workbook) and switch to the Visual Basic Editor.
2. In the Visual Basic Editor click the **Insert UserForm** button (or go to **Insert ➢UserForm**).
3. If the toolbox does not appear by itself (first click the form to make sure it isn't hiding) click the **Toolbox** button (or go to **View ➢Toolbox**).
4. To place a control on the form click the appropriate button on the toolbox then click the form. Controls can be moved by dragging them by their edges, or resized by dragging the buttons around their perimeter.
5. To edit the properties of a control, make sure the chosen control is selected then make the appropriate chang es in the **Properties** window. If you can't see the properties window go to **View ➢Properties Window**.
6. To remove a control from the form, select it and click the **Delete** key on your keyboard.

A UserForm will not actually do anything until the code that drives t he form and its various controls is created. The next step is to write the code that drives the form itself.

Adding the Code: 1 Initialising the Form

Initialising the Form:

Most forms need some kind of setting up when they open. This may be setting default values, making sure field are empty, or building the lists of combo boxes. This process is called *Initialising the Form* and it is taken care of by a macro called **UserForm_Initialize** (in case you are confused by my varying spelling of the word "initialis(z)e", it's because I speak English and VBA speaks American - but don't worry, VBA will spell it for you!). Here's how to build the code to initialise the Course Booking Form:

1. To view the form's code window go to **View ➢Code** or click **F7**.
2. When the code window first opens it contains an empty **UserForm_Click()** procedure. Use the drop-down lists at the top of the code window to choose **UserForm** and **Initialize**. This will create the procedure you need. You can now delete the UserForm_Click() procedure.
3. Enter the following code into the procedure:

```
Private Sub UserForm_Initialize()
    txtName.Value = ""
    txtPhone.Value = ""
    With cboDepartment
        .AddItem "Sales"
        .AddItem "Marketing"
        .AddItem "Administration"
        .AddItem "Design"
        .AddItem "Advertising"
        .AddItem "Dispatch"
        .AddItem "Transportation"
    End With
    cboDepartment.Value = ""
    With cboCourse
        .AddItem "Access"
        .AddItem "Excel"
        .AddItem "PowerPoint"
        .AddItem "Word"
        .AddItem "FrontPage"
    End With
    cboCourse.Value = ""
    optIntroduction = True
    chkLunch = False
    chkVegetarian = False
    txtName.SetFocus
End Sub
```

How the Initialise Code Works:

The purpose of the UserForm_Initialize() procedure is to pr epare the form for use, setting the default values for the various controls and creating the lists that the combo boxes will show.

These lines set the contents of the two text boxes to empty:

```
txtName.Value = ""
txtPhone.Value = ""
```

Next come the instructions for the combo boxes. First of all the contents of the list are specified, then the initial value of the combo box is set to empty.

```
With cboDepartment
    .AddItem "Sales"
    .AddItem "Marketing"
    (as many as necessary…)
End With
 cboDepartment.Value = ""
```

If required an initial choice can be made from the option group, in this case:

```
optIntroduction = True
```

Both check boxes are set to False (i.e. no tick). Set to True if you want the check box to appear already ticked:

```
chkLunch = False
chkVegetarian = False
```

Finally, The focus is taken to the first text box. This places the users cursor in the text box so that they do not need to click the box before they start to type:

```
txtName.SetFocus
```

Adding the Code: 2 Making the Buttons Work

There are three command buttons on the form and each must be powered by its own procedure. Starting with the simple ones…

Coding the Cancel Button:

Earlier, we used the Properties Window to set the **Cancel** property of the Cancel button to *True*. When you set the Cancel property of a command button to True, this has the effect of "clicking" that button when the user presses the **Esc** key on their keyboard. But this alone will not cause anything to happen to the form. You need to create the code for the click event of the button that will, in this case, close the form. Here's how:

1. With the UserForm open for editing in the Visual Basic Editor, double-click the Cancel button. The form's code window opens with the **cmdCancel_Click()** procedure ready for editing.

2. The code for closing a form is very simple. Add a line of code to the procedure so it looks like this:

```
Private Sub cmdCancel_Click()
    Unload Me
End Sub
```

Coding the Clear Form Button:

I added a button to clear the form in case the user wanted to change their mind and reset everything, and to make it easier if they had several bookings to make at one time. All it has to do is run the Initialise procedure again. A macro can be told to run another macro (or series of macros if necessary) by using the **Call** keyword:

1. Double-click the Clear Form button. The form's code window opens with the **cmdClearForm_Click()** procedure ready for editing.

2. Add a line of code to the procedure so it looks like this:

```
Private Sub cmdClearForm_Click()
    Call UserForm_Initialize
End Sub
```

Coding the OK Button:

This is the piece of code that has to do the job of transferring the user's choices and text input on to the worksheet. When we set the Cancel button's Cancel property to True we also set the OK button's **Default** property to *True*. This has of clicking the OK button when the user presses the **Enter** (or **Return**) key on their keyboard (providing they have not used their **Tab** key to tab to another button). Here's the code to make the button work:

1. Double-click the OK button. The form's code window opens with the **cmdOK_Click()** procedure ready for editing.

2. Edit the procedure to add the following code:

```
Private Sub cmdOK_Click()
    ActiveWorkbook.Sheets("Course Bookings").Activate
    Range("A1").Select
    Do
    If IsEmpty(ActiveCell) = False Then
        ActiveCell.Offset(1, 0).Select
    End If
    Loop Until IsEmpty(ActiveCell) = True
    ActiveCell.Value = txtName.Value
    ActiveCell.Offset(0, 1) = txtPhone.Value
    ActiveCell.Offset(0, 2) = cboDepartment.Value
    ActiveCell.Offset(0, 3) = cboCourse.Value
    If optIntroduction = True Then
        ActiveCell.Offset(0, 4).Value = "Intro"
    ElseIf optIntermediate = True Then
        ActiveCell.Offset(0, 4).Value = "Intermed"
    Else
        ActiveCell.Offset(0, 4).Value = "Adv"
    End If
    If chkLunch = True Then
        ActiveCell.Offset(0, 5).Value = "Yes"
    Else
        ActiveCell.Offset(0, 5).Value = "No"
    End If
    If chkVegetarian = True Then
        ActiveCell.Offset(0, 6).Value = "Yes"
    Else
        If chkLunch = False Then
        ActiveCell.Offset(0, 6).Value = ""
        Else
        ActiveCell.Offset(0, 6).Value = "No"
        End If
    End If
    Range("A1").Select
End Sub
```

How the CmdOK_Click code works:

The first two lines make sure that the correct workbook is active and moves the s election to cell A1:

```
ActiveWorkbook.Sheets("Course Bookings").Activate
Range("A1").Select
```

The next few lines moves the selection down the worksheet until it finds an empty cell:

```
Do
If IsEmpty(ActiveCell) = False Then
    ActiveCell.Offset(1, 0).Select
End If
Loop Until IsEmpty(ActiveCell) = True
```

The next four lines start to write the contents of the form on to the worksheet, using the active cell (which is in column A) as a reference and moving *along the row* a cell at a time:

```
ActiveCell.Value = txtName.Value
ActiveCell.Offset(0, 1) = txtPhone.Value
ActiveCell.Offset(0, 2) = cboDepartment.Value
ActiveCell.Offset(0, 3) = cboCourse.Value
```

Now we come to the option buttons. These have been placed in a frame on the form so the user can choose only one. An IF statement is used to instruct Excel what to for each option:

```
If optIntroduction = True Then
    ActiveCell.Offset(0, 4).Value = "Intro"
ElseIf optIntermediate = True Then
    ActiveCell.Offset(0, 4).Value = "Intermed"
Else
    ActiveCell.Offset(0, 4).Value = "Adv"
End If
```

VBA IF statements are much easier to manage than Excel's IF function. You can have as many options as you want, just insert an additional **ElseIf** for each one. If there were only two options, you wouldn't need the **ElseIf**, just the **If** and **Else** would suffice (don't forget - they all need an **End If**).

There is another IF statement for each check box. For the Lunch Required check box, a tick in the box means "Yes" the person requires lunch, and no tick means "No" they don't.

```
If chkLunch = True Then
    ActiveCell.Offset(0, 5).Value = "Yes"
Else
    ActiveCell.Offset(0, 5).Value = "No"
End If
```

We could use a similar IF statement for the Vegetarian check box, but if the person does not require lunch it is irrelevant whether or not they are vegetarian. I an y case, it would be wrong to assume that they were not vegetarian simply because they did not require lunch. The IF statement therefore contains a second, nested if statement:

```
If chkVegetarian = True Then
    ActiveCell.Offset(0, 6).Value = "Yes"
Else
    If chkLunch = False Then
        ActiveCell.Offset(0, 6).Value = ""
    Else
        ActiveCell.Offset(0, 6).Value = "No"
    End If
End If
```

A tick in the box means "Yes" the person is vegetarian. If there is no tick in the box, the nested IF statement loo ks at the Lunch Required check box. If the Lunch Required check box has a tick in it then no tick in the Vegetarian check box means that the person is not vegetarian so it inserts "No" into the cell. However, if the Lunch Required check box does not have a tick in it, then we do not know whether or not the person is vegetarian (it doesn't matter anyway) so the cell is left blank ("").

Finally the selection is taken back to the beginning of the worksheet, ready for the next entry:

```
Range("A1").Select
```

Adding the Code 3: Manipulating the Form

Finally, an example of how the controls on a form can be manipulated whilst it is in use. When the control properties were set, the **Enabled** property of the Vegetarian check box was set to *False*. When a control is not enable d the *user* cannot enter a value into it, although it can hold a value that was there already, and VBA can add, remove or change the value.

We don't need to know whether or not the person is vegetarian (even if they are!) if they aren't ordering lunch. So, the Vegetarian check box remains disabled unless a tick is placed in the Lunch Required check box. Then the user is free to tick the Vegetarian check box if they want to. If they tick it we will know that they have answered "Yes" and if they don't we know they have answered "No".

We can toggle the **Enabled** property from *False* to *True* by having a procedure that runs automatically whenever the value of the Lunch Required check box changes. Fortunately, more controls have a *Change* procedure and the one we use here is **chkLunch_Change()**. We'll use this to enable the Vegetarian check box when the Lunch Required check box is ticked, and disable it when the Lunch Required check box is not ticked.

There's just one more thing we need to do. Supposing someone ticked th e Lunch Required check box, and also ticked the Vegetarian check box. Then they changed their mind and removed the tick from the Lunch Required check box. The Vegetarian check box would be disabled but the tick that was put in earlier would remain.

An extra line of code can make sure the tick is removed when the box is disabled. Here's the whole thing:

```
Private Sub chkLunch_Change()
    If chkLunch = True Then
        chkVegetarian.Enabled = True
    Else
        chkVegetarian.Enabled = False
        chkVegetarian = False
    End If
End Sub
```

Opening the Form

The form is now ready for use so it needs to be opened with a simple macro. That can be attached to a custom toolbar button, a command button drawn on the worksheet, or any graphic (right click the graphi c and choose **Assign Macro**). If necessary, create a new module for the workbook and add this procedure:

```
Sub OpenCourseBookingForm()
    frmCourseBooking.Show
End Sub
```