Build a UserForm for Excel

Introduction

A UserForm is a custom-built dialog box that you build using the Visual Basic Editor. Whilst this example works in Excel you can use the same techniques to create a UserForm in any of the Microsoft Office programs that support VBA.

With a UserForm you can create a user-friendly interface for your workbook or document, making data entry more controllable for you and easier for the user.

About the Project

This document shows you how to build a simple UserForm for entering personal expenses data on to a worksheet in Excel. The work is divided into two main sections: building the form itself and then writing the VBA code to make it work. The finished UserForm will look something like this (*Fig. 1*).

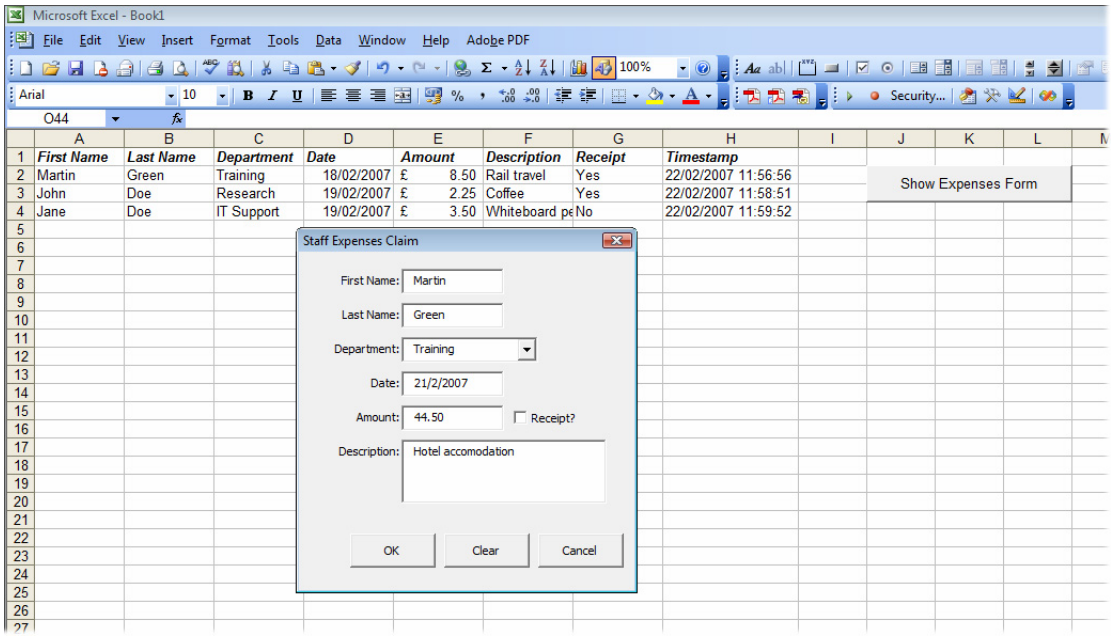

Fig. 1 The finished UserForm project.

NOTE: The screenshots here show how things look in Excel 2003 running on Windows Vista. If you are working in an different version of Excel or Windows the appearance will be slightly different but all the techniques explained here will be exactly the same.

Build the Form

Start Excel and open the Visual Basic Editor (keyboard shortcut: **[Alt]+[F11]**). You will need to use both the **Project Explorer** and the **Properties Window** so if they are not visible open them from the **View** menu.

HINT: When building your UserForm try to avoid double-clicking on anything (unless the instructions tell you to do so) because this sometimes opens the form's code window. If you do this accidentally simply close the code window by clicking its Close button, or switch back to the UserForm design window with the keyboard shortcut **[Control]+[Tab]**.

Insert a New UserForm

Make sure that the current workbook (e.g. *VBAProject (Book1)*) is selected in the Project Explorer then open the **Insert Menu** and choose **UserForm**. When you do this a new, blank UserForm appears in the code window of the Visual Basic Editor and a corresponding entry appears in the Project Explorer (*Fig. 2*). The Project Explorer shows a new folder named *Forms* containing the new UserForm which has been given the name *UserForm1*.

You should also see the **Toolbox** (*Fig. 3*). If it is not visible click anywhere on the new UserForm (the Visual Basic Editor hides the toolbox when it thinks you are working elsewhere) and if it still does not appear open it from the **View** menu.

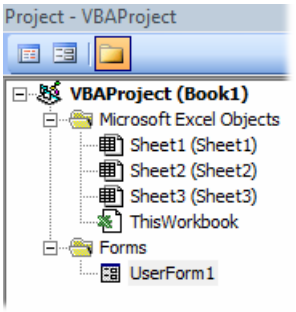

Fig. 2 The Project Explorer shows the UserForm.

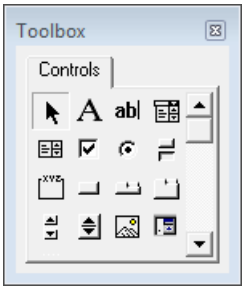

Fig. 3 The Toolbox

The UserForm has a dotted border around it. On the border, in the lower-right corner and halfway along the bottom and right sides of the form, are small white squares. These are the resizing handles. You can use the mouse to drag these handles to make the UserForm the required size. The grid of dots on the form is to help you easily align items you place there.

Rename the UserForm and Add a Caption

A single project can include many UserForms so it is a good idea to give each one a meaningful name. With the UserForm selected find the **Name** property in the Properties Window (it is normally the first item in the list) and change it *frmExpenses*. Then change the **Caption** property to *Staff Expenses Claim*. The Project Explorer now displays the UserForm's new name and the Title Bar of the form immediately changes to show the new caption (*Fig. 4*).

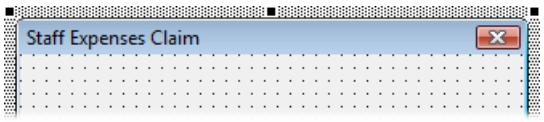

Fig. 4 The form's Title Bar shows the new caption.

When naming forms and their controls remember that you must not include spaces in the name, or use any of the VBA "reserved words" (i.e. those keywords that are part of the VBA language such as "Date").

Add a TextBox Control and a Label

The *controls* are the items, such as textboxes, comboboxes and command buttons, that will be placed on the form. The standard selection of controls is represented by buttons on the Toolbox. Point at a Toolbox button to see a tooltip showing the name of the control.

Add a TextBox control to the form by clicking on the **TextBox** button in the Toolbox then clicking somewhere near the centre of the form. As with the UserForm itself any control that you place on the form shows a dotted border and resizing handles when the item is selected (click on any item to select it).

You can change the size and shape of a control either by dragging the resizing handles (the mouse pointer will change to a double-headed arrow (*Fig. 5*)) or by changing the values of its *Height* and *Width* properties in the Properties Window. To move a control drag the dotted border at a point between resizing handles (the mouse pointer will show a four-headed arrow (*Fig. 6*)) or change the values of its *Top* and *Left* properties.

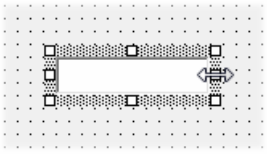

Fig. 5 Resizing a control.

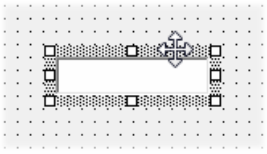

Fig. 6 Moving a control.

Drag the textbox to a point near the top of the UserForm and about halfway across the form.

Each control should have a meaningful name so that when you write the code you can easily identify it. This one currently has the name *TextBox1*. Use the Properties Window to change its name to *txtFirstName*.

HINT: It is helpful when naming controls to add a prefix describing the type of control ("txt" for textbox, "cbo" for combobox etc.). This reminds you what type of control it is when you are working in the code. It forces the names to appear together when displayed in a list. It also lets you use words that are otherwise reserved (e.g. txtDate *instead of* Date*).*

Now use the toolbox to place a **Label** control on the form. To change the caption of the label you can either type directly on to the label or you can change its *Caption* property in the Properties Window. Change the label's caption to *First Name:*.

Change the *TextAlign* property of the label to *3-fmTextAlignRight* then double-click the lower-right corner resizing handle to snap the label to fit the text (*Fig. 7*). Drag the label to a position just to the left of the *FirstName* textbox.

Fig. 7 Double-click the lower-right corner handle to snap the label to size.

When you move controls around by dragging them they snap to points on the grid. Whilst this is a useful feature, sometimes you need to position objects more accurately. You will notice that you can not place the label exactly level with the centre of the textbox. The grid forces it to be too high or too low. Use the Properties Window to subtract (or add as necessary) 3 units from the *Top* property of the label so that it is correctly positioned in relation to the textbox (*Fig. 8*).

Fig. 8 Use the Properties Window to finely adjust the position of a control.

It isn't necessary to give the label a different name because, in this project, you will not be referring to it in the code, although in other circumstances you might want to do so.

Add the Remaining Controls

Use the same techniques to add the remaining controls to the form. You need to add four textboxes, a combobox (a textbox with a drop-down list), a checkbox and three command buttons. Here is a list of the remaining controls you need to add and their properties:

TextBox ..... Name: *txtLastName*
Label..... Caption: *Last Name:*
ComboBox ..... Name: *cboDepartment*
Label..... Caption: *Department:*
TextBox ..... Name: *txtDate*
Label..... Caption: *Date:*
TextBox ..... Name: *txtAmount*
Label..... Caption: *Amount:*
CheckBox..... Name: *chkReceipt*, Caption: *Receipt?*
TextBox ..... Name: *txtDescription*, Height: *45*, Width: *132*, Scrollbars: *2-fmScrollbarsVertical*
Label..... Caption: *Description:*
CommandButton ... Name: *cmdOK*, Caption: *OK*
CommandButton ... Name: *cmdClear*, Caption: *Clear*
CommandButton ... Name: *cmdCancel*, Caption: *Cancel*

*HINT: At any time you can check out exactly how the UserForm will look in use by pressing the **[F5]** key on your keyboard or clicking the **Run** button on the Visual Basic Editor toolbar. Doing this will open the UserForm in its host program (in this case Excel). To return to the Visual Basic Editor, close the UserForm by clicking the close button **[x]** in its upper-right corner.*

The finished UserForm should look something like this (*Fig. 9*):

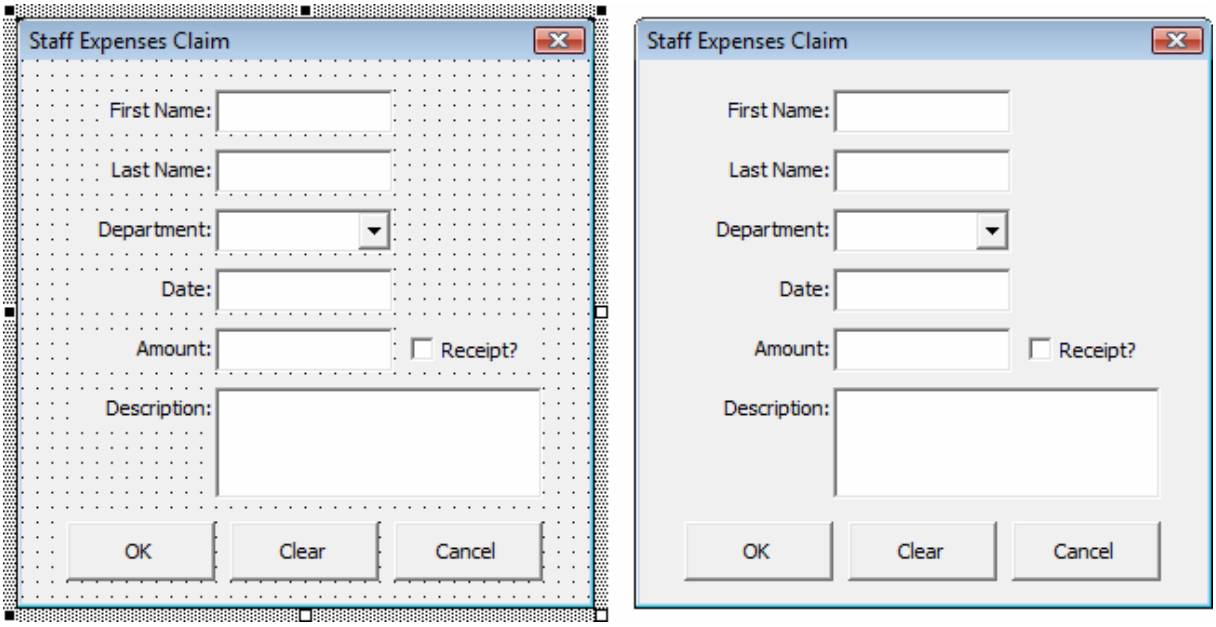

Fig. 9 The finished UserForm in design view (left) and in use (right).

Create the ComboBox List

The *Department* combobox now needs to be told where to get the information to build its list. There are two ways to do this. It can be done with code (this method is described in the next section) or it can refer to a named range of cells in the workbook. The latter method is often preferred because you can easily edit the list without having to rewrite any code.

Switch to Excel and open a worksheet in the same workbook. Type a column of items representing the entries you want to appear in the combobox list. Put one item in each cell. If you want the items to appear in alphabetical order sort the list in the worksheet.

Now select the cells containing the list items and name the range of cells. The easiest way is to click in the *Name* box (the box just above and to the left of cell A1), type the name *Departments* (*Fig. 10*) then press **[Enter]**. Click somewhere else on the worksheet then check that you have correctly named the range by clicking the down-facing arrow to the right of the Name box. You should see your range name in the list. Choose it and check that Excel selects the cells that contain your list.

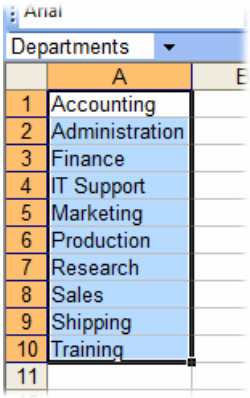

Fig. 10 Name a range of cells containing the list.

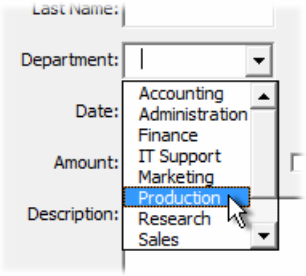

Fig. 11 The combobox displays the list.

If you add items to the list at a later date you may need to redefine the list. You can do this by opening Excel's **Insert** menu and choosing **Name** then **Define**.

Return to the Visual Basic Editor and click on the *Department* combobox to select it then go to the Properties Window and find the *RowSource* property. Enter the same name as you used for the range containing your list (in this example *Departments*).

Test the form (press the **[F5]** key) and see that the combobox now displays your list (*Fig. 11*).

Check the Tab Order

Many people when working in a form like to move around from control to control by clicking the **[Tab]** key on their keyboard. The order in which the tab key moves you around a form is initially defined by the order in which you placed the controls on the form.

Run the form (open it in Excel) and, starting from the *FirstName* textbox, press the **[Tab]** key repeatedly and check that it takes you through the form in a logical order. If you want to change the order close the form and in the Visual Basic Editor open the **View** menu and choose **Tab Order**. Here you can move items up and down the list to control the behaviour of the **[Tab]** key in the form (*Fig. 12*).

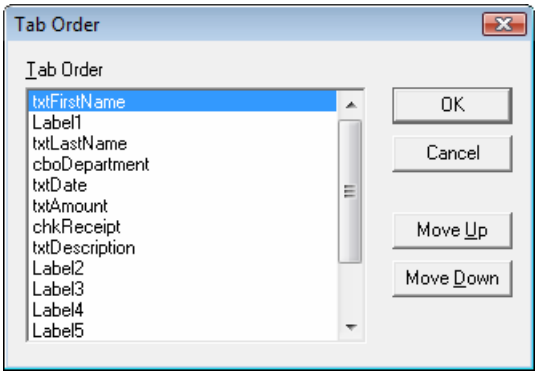

Fig. 12 The Tab Order dialog box.

Write the VBA Code

The design of the form is now finished. The next job is to write the VBA code to power it. The code is needed to power the three command buttons…

Coding the Cancel Button

The *Cancel* button is the simplest one to code. It needs to do the same job as the built-in close button (**[x]**) in the upper right corner of the form.

Double-click the *cmdClose* command button (or right-click it and choose **View Code**) to open the UserForm's code module. The Visual Basic Editor will have written the *Sub* and *End Sub* lines of the button's *Click* event for you. Place your cursor in the empty line between these lines, press your **[Tab]** key then enter the line:

```
Unload Me
```

Your code should look like this (*Listing 1*):

Listing 1

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

Test the code. Open the **Debug** menu and choose **Compile VBAProject**. If you get an error message check your typing and compile again. Then switch to the form design window (press **[Control]+[Tab]** or double click the form's name in the Project Explorer). Press the **[F5]** key to run the form in Excel. Click the form's **Cancel** button and the form should close and return you to the Visual Basic Editor.

Coding the OK Button

The *OK* button has three jobs to do. It has to:

1. Check the user's input so that all the required information has been supplied (this is called "validation".
2. Write the data on to the worksheet in Excel.
3. Clear the form ready for the next entry.

In the design view of the form double-click the *cmdOK* button and enter the following lines into the *cmdOK_Click* event procedure (*Listing 2*):

Listing 2

```
Private Sub cmdOK_Click()
    If Me.txtFirstName.Value = "" Then
        MsgBox "Please enter a First Name.", vbExclamation, "Staff Expenses"
        Me.txtFirstName.SetFocus
        Exit Sub
    End If

End Sub
```

This procedure uses an *If Statement* to check the contents of the *txtFirstName* textbox. If the textbox is empty (i.e. its contents are "" – the two quote marks with nothing between them represents "nothing") a message is displayed, the focus is set to that textbox (the user's cursor is taken there), and the procedure is cancelled.

As before, compile and test the code. Open the form and, without entering anything in the *First Name* text box, click the **OK** button. You should see the error message (*Fig. 13*). Dismiss the message box then type an entry in the *First Name* textbox and try again. No message box should be displayed when you click the **OK** button.

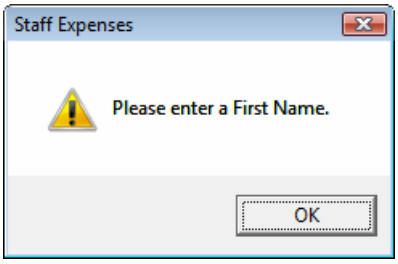

Fig. 13 The error message reminds the user to enter a First Name.

Make a similar entry for each textbox and for the combobox. You can view the complete code listing at the end of this document (*Listing 8*).

In addition to checking that an entry is present, it is also sometimes necessary to check that the entry is correct. Enter the following statements (*Listing 3*). They use the *IsNumeric()* function to check that the value in the *txtAmount* textbox is a number (and not something else such as text); and the *IsDate()* function to check that the value in the *txtDate* textbox is a date.

Listing 3

```
If Not IsNumeric(Me.txtAmount.Value) Then
    MsgBox "The Amount box must contain a number.", vbExclamation, "Staff Expenses"
    Me.txtAmount.SetFocus
    Exit Sub
End If
If Not IsDate(Me.txtDate.Value) Then
    MsgBox "The Date box must contain a date.", vbExclamation, "Staff Expenses"
    Me.txtDate.SetFocus
    Exit Sub
End If
```

Now, having reached a point where all the required entries are present and correct, it's time to write the entries on to the worksheet. This code involves using a *variable* to hold the number of rows of data on the worksheet.

Make an empty line at the top of the current procedure, immediately after the statement *Private Sub cmdOK_Click()* and enter the line:

```
Dim RowCount As Long
```

I have assumed that the entries are going to be made on *Sheet1* of the current workbook, starting in cell *A1*. You might like to prepare the worksheet by typing a row of headings in the top row. The code will work the same way if there are headings or not.

Return to the end of your code and enter the new line:

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

This statement counts how many rows of data are included in the region that includes cell *A1* and stores that number in the *RowCount* variable. Now enter the lines that write the date on to the worksheet (*Listing 4*):

Listing 4

```
With Worksheets("Sheet1").Range("A1")
    .Offset(RowCount, 0).Value = Me.txtFirstName.Value
    .Offset(RowCount, 1).Value = Me.txtLastName.Value
    .Offset(RowCount, 2).Value = Me.cboDepartment.Value
    .Offset(RowCount, 3).Value = DateValue(Me.txtDate.Value)
    .Offset(RowCount, 4).Value = Me.txtAmount.Value
    .Offset(RowCount, 5).Value = Me.txtDescription.Value
    If Me.chkReceipt.Value = True Then
        .Offset(RowCount, 6).Value = "Yes"
    Else
        .Offset(RowCount, 6).Value = "No"
    End If
    .Offset(RowCount, 7).Value = Format(Now, "dd/mm/yyyy hh:nn:ss")
End With
```

The code uses a number of similar statements to write the value of each control into a cell. Each cell is identified by its position relative to cell *A1* by using the VBA *Offset* property. This requires two numbers, the first representing the number of rows away from cell *A1* (which is held in the *RowCount* variable), the second representing the number of columns away from cell *A1* (which is written into the code as a number).

Note that the *DateValue()* function is used to change the date entry into a real date (rather than a date represented as text) before passing it to Excel.

The value of a checkbox is expressed as "TRUE" or "FALSE" so, because I wanted to see "Yes" or "No" on the worksheet, the code uses an *If Statement* to make the required entry.

Finally a timestamp is written into the last column using the *Format()* function to specify how the exact time, as supplied by the *Now()* function, is displayed. Now is a good time to compile and test the code again.

To complete the procedure, after the data has been written to the worksheet, the form needs to be emptied. This requires another variable to be entered at the top of the procedure:

```
Dim ctl As Control
```

This variable represents the controls on the worksheet and will be used in the following loop which visits each control, checks to see if it is a textbox or a combobox, and if it is it sets the control's value to an empty string (""). If the control is a checkbox it sets its value to *False* Enter the following lines (*Listing 5*):

Listing 5

```
For Each ctl In Me.Controls
    If TypeName(ctl) = "TextBox" Or TypeName(ctl) = "ComboBox" Then
        ctl.Value = ""
    ElseIf TypeName(ctl) = "CheckBox" Then
        ctl.Value = False
    End If
Next ctl
```

Compile and test the code again. If any errors occur then check that your typing is exactly as shown here.

Coding the Clear Button

The function of this button is to clear the form manually if the user wishes to do so. It uses exactly the same procedure as the last part of the *OK* button's procedure, so double-click the *cmdClear* button to create its click event procedure and enter the code shown in *Listing 5*. You can copy the code from the *OK* button's procedure to save time.

Compile, Test and Save the Finished UserForm

That completes the coding of the form. Compile and test the code. If you are satisfied that the form is working correctly save the file. The job is almost finished. All that remains is to create a macro to open the form.

A Macro to Open the UserForm

As you have seen, it is easy to open the UserForm from the Visual Basic Editor, but the person who is going to use this tool needs an easy way to open the form from Excel. There are several ways to do this, all involving a macro containing the very simple statement:

```
frmExpenses.Show
```

Manually Opening the Form

This statement can be included in a macro that the user can call from the usual menu. To create this macro first go to the Visual Basic Editor's **Insert** menu and choose **Module** to add a standard module to the workbook containing the UserForm. Enter the following code into the new module's code window (*Listing 6*):

Listing 6

```
Sub OpenExpensesForm()
    Worksheets("Sheet1").Activate
    frmExpenses.Show
End Sub
```

The first line is optional. It tells Excel to switch to *Sheet1*. But, since the code that writes the data on to the worksheet specifies the worksheet by name, it could be omitted and the data would still be written in the correct place.

The user can run this macro from the menu in the usual way (**Tools > Macro > Macros**) or you could assign it to a custom menu item, toolbar button, a button on the worksheet or a drawing object.

Opening the Form Automatically

You can make use of one of Excel's built-in event procedures to open the UserForm automatically when the workbook is opened. In the Visual Basic Editor locate and double-click the *ThisWorkbook* module in the Project Explorer. This module exists to hold macros specific to the workbook itself.

At the top of the code window there are two drop-down lists. The left-hand one will currently read *General*. Open the list and choose **Workbook**. The Visual Basic Editor automatically creates the *Workbook_Open* macro for you. Any code you place in this macro will be executed automatically when the workbook opens. (If you want to see what else you can do here take a look at the other items on the right-hand list.) Complete the macro code as follows (*Listing 7*):

Listing 7

```
Private Sub Workbook_Open()
    Worksheets("Sheet1").Activate
    frmExpenses.Show
End Sub
```

Complete Code Listing for the UserForm

Here is a complete listing of the code in the UserForm's code module (*Listing 8*)

Listing 8

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

Private Sub cmdClear_Click()
' Clear the form
    For Each ctl In Me.Controls
        If TypeName(ctl) = "TextBox" Or TypeName(ctl) = "ComboBox" Then
            ctl.Value = ""
        ElseIf TypeName(ctl) = "CheckBox" Then
            ctl.Value = False
        End If
    Next ctl
End Sub

Private Sub cmdOK_Click()
```

```
    Dim RowCount As Long
    Dim ctl As Control
' Check user input
    If Me.txtFirstName.Value = "" Then
        MsgBox "Please enter a First Name.", vbExclamation, "Staff Expenses"
        Me.txtFirstName.SetFocus
        Exit Sub
    End If
    If Me.txtLastName.Value = "" Then
        MsgBox "Please enter a Last Name.", vbExclamation, "Staff Expenses"
        Me.txtFirstName.SetFocus
        Exit Sub
    End If
    If Me.cboDepartment.Value = "" Then
        MsgBox "Please choose a Department.", vbExclamation, "Staff Expenses"
        Me.txtFirstName.SetFocus
        Exit Sub
    End If
    If Me.txtDate.Value = "" Then
        MsgBox "Please enter a Date.", vbExclamation, "Staff Expenses"
        Me.txtFirstName.SetFocus
        Exit Sub
    End If
    If Me.txtAmount.Value = "" Then
        MsgBox "Please enter an Amount.", vbExclamation, "Staff Expenses"
        Me.txtFirstName.SetFocus
        Exit Sub
    End If
    If Me.txtDescription.Value = "" Then
        MsgBox "Please enter a Description.", vbExclamation, "Staff Expenses"
        Me.txtFirstName.SetFocus
        Exit Sub
    End If
    If Not IsNumeric(Me.txtAmount.Value) Then
        MsgBox "The Amount box must contain a number.", vbExclamation, "Staff Expenses"
        Me.txtAmount.SetFocus
        Exit Sub
    End If
    If Not IsDate(Me.txtDate.Value) Then
        MsgBox "The Date box must contain a date.", vbExclamation, "Staff Expenses"
        Me.txtDate.SetFocus
        Exit Sub
    End If
' Write data to worksheet
    RowCount = Worksheets("Sheet1").Range("A1").CurrentRegion.Rows.Count
    With Worksheets("Sheet1").Range("A1")
        .Offset(RowCount, 0).Value = Me.txtFirstName.Value
        .Offset(RowCount, 1).Value = Me.txtLastName.Value
        .Offset(RowCount, 2).Value = Me.cboDepartment.Value
        .Offset(RowCount, 3).Value = DateValue(Me.txtDate.Value)
        .Offset(RowCount, 4).Value = Me.txtAmount.Value
        .Offset(RowCount, 5).Value = Me.txtDescription.Value
        .Offset(RowCount, 6).Value = Format(Now, "dd/mm/yyyy hh:nn:ss")
        If Me.chkReceipt.Value = True Then
            .Offset(RowCount, 7).Value = "Yes"
        Else
            .Offset(RowCount, 7).Value = "No"
        End If
    End With
' Clear the form
    For Each ctl In Me.Controls
        If TypeName(ctl) = "TextBox" Or TypeName(ctl) = "ComboBox" Then
            ctl.Value = ""
        ElseIf TypeName(ctl) = "CheckBox" Then
            ctl.Value = False
        End If
    Next ctl
End Sub
```