VBA5-Introduction to User Forms

You can communicate with a user with MSGBOX and use the INPUT statement to request a value. If you want to get several values or display several values at a time, you'll want to create a User-Defined Form. This session will introduce forms: how to create a form, accept and store data, and display and hide forms. You'll learn the code for looping until the form has good data or the user cancels it. You'll also get an introduction to the various controls you can put on a form, such as command buttons and lists.

Constants

1. Variables are containers that hold values during process.
2. These values can change over the course of processing.
3. A Constant is like a variable, but you assign it a value at the start of processing and the value cannot change during processing.
4. Define a Constant using:

```
Constant [ConstantName] as [DataType] = Value
```

5. Constants are useful in many ways. Some examples:

```
Constant intLastCol as Integer = 27
```
A constant that has the number of the last column in your list.

```
Constant intNameCol as Integer = 3
```
A constant that defines the Name column as column 3

```
Constant strCompanyName as String = "Pioneer Training, Inc."
```
'A constant that holds the name of the company.

6. Defining your fixed rows and columns as Constants is far better than using the row or column number. If you change a column, you can change the constant assignment once and all your processing will be updated. If you use a number, you will have to search for every instance of that number in your code.
7. Both Visual Basic and Visual Basic for Applications for Excel have additional constants built in. You can use the underlying number assigned to the constant, but it is far better to use the constant.
    a. Visual Basic constants begin with **vb**, as in **vbYes** or **vbNo** or **vbCrLf**.
    b. Excel constants begin with **xl**, as in the statement:

```
Selection.PasteSpecial Paste:=xlPasteValues,
Operation:=xlNone, SkipBlanks _
      :=False, Transpose:=False
```

8. If Microsoft changes the underlying number, your code is not affected. Also, your code is much more readable:
   If intButtonPressed = 6 Then MsgBox("You pressed Yes") or

```
If intButtonPresssed = vbYes Then MsgBox("You pressed Yes")
```

Communicating with Your Users

1. While you can create full forms in VBA, you can often use simpler methods to communicate with your user.
2. To display a message to your user, you can use a Message Box.
3. To receive input from your user, you can use an Input Box.

MsgBox

1. The `MsgBox` statement is a simple way to communicate.
2. You use the format `MsgBox ("Message")` to display your message:
    a. Processing stops.
    b. The user sees the message.
    c. The use can respond by clicking OK or using the X to close the dialog.
3. You can also use the `MsgBox` as a function by adding additional options.
    a. The `MsgBox` function returns the value of the button that is clicked.
    b. You have control over the message, dialog box title, icon, buttons, and help file for the dialog.
    c. Only the message is required. Use commas to separate your values
    d. Each button returns an integer value corresponding to the button pressed. You can test this return value and process accordingly.
    e. The full definition of the MsgBox function is:

```
intReturnValue = MsgBox("Message", _
      [buttons],[Title],[HelpFile],[Context]
```

    f. The Buttons include which buttons to show, the icon to show, modality, and more.
    g. Modality is the term for whether the dialog box will require user input before she can continue (Modal) or whether she can work with Excel, etc. while the dialog box is open. (Modal not set).
    File |Save As is modal. Find is not modal.
    h. You combine buttons and icons:

```
intReturnValue=MsgBox("Do you want to _
      continue?",vbYesNo+vbInformation, _
      "Continue Processing")
```

4. Use an `If Then Else` or `Select Case` statement to evaluate the results and process accordingly.

EXERCISE: Hello World

1. Display "Hello World" in a Message Box.
2. Add a Title and Yes, No, Cancel buttons to the dialog.
3. Add a Question Mark icon to the dialog.

InputBox

1. The **InputBox** statement lets you ask a user for information and receive one piece of information back.
2. The information is returned as a string.
3. **InputBox** is a function and has similar values as the MsgBox function:

```
InputBox("Message",[Title] ,[DefaultValue] [,XPosition] _
      [, YPostion] [, HelpFile, Context])
intReturnString=InputBox("Please enter your first name", _
      "Accept User Name","User")
```

4. The return value is blank if the user presses Cancel or the X in the title bar.
5. You should test the return value for an acceptable value. (i.e., name not blank, number not below 0 or a text value, etc.).
6. You can use the return value in your processing or add it directly to a cell using Cells or Range.

EXERCISE: Get the user's name

1. Get the user's name from an Input Box.
2. Use a default of "User Name" in the dialog.
3. Welcome the user with a Message Box.

Creating a Form

1. Input boxes are useful, but often you need to ask a user for a number of different pieces of information or display a number of pieces of information.
2. In this case, you use a User Form.
3. The User Form creates a dialog box that can display text and accept input from a text field, a list, a combo box, a command button, radio buttons, check boxes and more.
4. The VBA Editor lets you create and modify a User Form.
5. Use the Toolbox and the Properties window to customize a form.
6. You should always put a way to close the form and cancel the form on the form.
7. The process of using a form is:
    a. Load the form
    b. Show the form
    c. Let the user interact with the form
    d. Hide the form
    e. Unload the form
8. Forms need to be loaded and unloaded into memory. Unloading a form wipes all user input. Hiding a form keeps the last entries in the form which show when you show it.
9. Name a form using the initial letters **frm**, as in **frmWelcomeUser** or **frmDisplayBalance**.

EXERCISE: Create a User Form

1. Create a User Form.
2. Name it frmFirstForm.
3. Add an OK button and a Cancel button to the form.
4. Edit the display text to display Cancel and OK.
5. Change the name of the buttons to btnCancel and btnOK.

Events that Trigger Specific Code on a Form Object

1. In an object-oriented programming language, each object has programming code associated with different user input attached to it.
    a. For example, a button that you add to a form can have code associated with the user clicking the OK or Cancel buttons.
    b. Each action the user can perform is called an Event.
    c. When the user clicks the button, the Click Event code is executed.
2. There are many events possible: click, button down, button up, hover over, right-click, etc. Each of these events can have different code attached.
3. To add code to an object, right-click on the object and select View Code.
    a. Excel will display the Code Window for the object, with a Subroutine for the default event.
    b. For a button, the default event is Click.

EXERCISE: Add a Click Event to Your Buttons

1. Add a Click event to the Cancel button that hides the form.
2. Add a Click event to the OK button that hides the form.

Displaying and Hiding a Form

1. In order to use a form, you must first load it. This places the form into memory.

```
Load [FormName]
```

2. Next, you need to Show the form. This actually displays it in Excel.

```
[FormName].Show
```

3. When you are done processing, typically with an OK or Cancel button, you first hide the form.

```
[FormName].Hide
```

4. When you are done with the form, unload it from memory. This wipes all user input from the form.

```
Unload [FormName]
```

EXERCISE: Display Your User Form

1. Create a Subroutine that loads, displays, and unloads your form. (The buttons should hide it).
2. SAVE YOUR WORK.
3. Test your form.

Labels and Text Boxes

1. The simplest way to get user input from a User Form is with a Text box.
2. The Text Box displays a field into which the user can enter a value.
3. The value is a string, but it can be converted as necessary.
4. You need to add a Label to the field so the user knows what the Text box requires.
5. Prefix a label with `lbl` and a Text box with `txt` or `tbx`.
6. For clarity, use the same name for the label and the Text box, i.e. `lblFirstName` and `txtFirstName`.

EXERCISE: Add Two Text Boxes

1. Add a text box and a label for the user's Last Name.
2. Add a text box and a label for the Amount of the Sale.
3. Test your dialog.

Storing User Input into a Cell

1. There are two ways to store a value from user input.
2. The easiest is to place the input into a cell.
3. In the OK button code, take the value in the Text box and assign it to a cell.
4. It is a good idea to test first for good information.
5. The format of the Text box value is:

   [FormName].[TextBoxName].Value

   a. You can assign this to a cell:

```
Cells(2,1).Value = frmFirstForm.txtLastName.Value
```

   b. You can read a value into the text box to initialize it:

```
frmFirstForm.txtLastName.Value = Cells(2,1).Value
```

Storing User Input into a Public Variable

1. If you want to process a value before adding it to a cell, you can load the value into a variable.
2. Variables only retain their values while that particular subroutine is running so if you store a value into a variable in the form, it will vanish once you close the form.
   a. This is called the Scope of the variable.

3. By defining a variable as a Public variable, it holds its value as long as the code is running.
4. You define a Public variable at the top of a module with the keyword Public:

```
Public [VariableName] as [DataType]
```

5. In a User Form, store the value of a Text box (or other input object) into a Public variable and you can use it when the form is closed.
6. You can use one of the conversion functions in VBA to convert any type of data into any other.
    a. The full set is available in the VBA help.
    b. The basic format is `Ctype([Variable])`:
    `CInt` converts a string to an Integer
    `CDbl` converts a string to a Double
    `CCur` converts a string to a Currency value

```
intNumber = CInt(strNumberAsText)
curAmount = CCur(strNumberAsText)
```

EXERCISE: Using a Public Variable

1. Define a Public variable called curSale.
2. Test for a blank value.
3. Convert the data into Currency before assigning it.
4. Assign the value of the Sale Amount to the curSale variable.

Looping until Good Data

1. You should always test your input forms for good data and be able to deal with bad input.
2. The more general your audience, the more bullet-proof your code needs to be.
3. The easiest way to test for good data is with a loop:
    a. Set two public variables, `blnGoodData` to False and `blnCancel` to False.
    b. Add a `While` loop that loops while `blnGoodData = False` and `blnCancel = False`.
    c. Set `blnGoodData` to True
    d. Load the form and show it.
    e. In the OK button, test each input field for acceptable data.
    f. If any of the fields has invalid data, set `blnGoodData` to `False`. You may want additional code to either blank the bad data, keep track of which fields are bad, etc.
    g. At the end of the tests, hide the form.
    h. `If blnCancel = True,` then the user clicked Cancel and process that.
    i. `If blnGoodData = False`, display a message. The loop will continue until the user enters all good data or clicks Cancel.
    j. In the Cancel button code, set the Click event to `blnCancel = True` and hide the form.
    k. Here is a sample loop in a Module:

```
Public blnGoodData As Boolean
Public blnCancel As Boolean

Sub LoopUntilGoodData()
blnGoodData = False
blnCancel = False
Range("A1:B2").ClearContents

Load frmLoopUntilGood
While blnGoodData = False And blnCancel = False
   blnGoodData = True
   frmLoopUntilGood.Show

   If blnGoodData = False Then
        MsgBox ("Invalid Data. Please re-enter or click
Cancel.")
   End If
Wend
Unload frmLoopUntilGood
End Sub
```

l. Here is the corresponding code in the form:

```
Private Sub cmdCancel_Click()
   blnCancel = True
   frmLoopUntilGood.Hide
End Sub

Private Sub cmdOK_Click()
   If frmLoopUntilGood.txtFirstName = "" Then
        blnGoodData = False
   Else
        Range("A1").Value = "First Name"
        Range("B1").Value =
frmLoopUntilGood.txtFirstName.Value
        Range("A1:B1").EntireColumn.AutoFit

   End If
   If frmLoopUntilGood.txtLastName = "" Then
        blnGoodData = False
   Else
        Range("A2").Value = "Last Name"
        Range("B2").Value = frmLoopUntilGood.txtLastName.Value
        Range("A1:B1").EntireColumn.AutoFit
   End If
   frmLoopUntilGood.Hide
End Sub
```

Lists and Combo Boxes

1. In addition to Text boxes, other common controls are Lists and Combo Boxes.
2. A List box will display a list of items for the user to choose.
3. A Combo box will display a list of items and also allow the user to enter her own text.
4. To initialize a list or combo box, you load the control with your choices.
5. In OK button processing, you read the value of the control into a public variable or a cell.
6. Here is the Module code that loads and saves a combo box field:

```
Sub UsingComboBoxes()
      blnGoodData = False
      blnCancel = False

      Range("A1:B3").ClearContents

      frmAddComboBox.cboState.AddItem "MA"
      frmAddComboBox.cboState.AddItem "ME"
      frmAddComboBox.cboState.AddItem "VT"
      frmAddComboBox.cboState.AddItem "NH"
      frmAddComboBox.cboState.AddItem "CT"

      Load frmAddComboBox

      While blnGoodData = False And blnCancel = False
            blnGoodData = True
            frmAddComboBox.Show

            If blnGoodData = False Then
                  MsgBox ("Invalid Data. Please re-enter or click
Cancel.")
            End If

      Wend
      Unload frmAddComboBox
End Sub
```

7. Here is the corresponding Code for the form:

```
Private Sub cmdCancel_Click()
      blnCancel = True
      frmLoopUntilGood.Hide
End Sub

Private Sub cmdOK_Click()
      If frmAddComboBox.txtFirstName = "" Then
            blnGoodData = False
      Else
            Range("A1").Value = "First Name"
            Range("B1").Value = frmAddComboBox.txtFirstName.Value
            Range("A1:B1").EntireColumn.AutoFit

      End If

      If frmAddComboBox.txtLastName = "" Then
            blnGoodData = False
      Else
            Range("A2").Value = "Last Name"
            Range("B2").Value = frmAddComboBox.txtLastName.Value
            Range("A1:B1").EntireColumn.AutoFit
      End If

      If frmAddComboBox.cboState.Value = "" Then
            blnGoodData = False
      Else
            Range("A3").Value = "State"
            Range("B3").Value = frmAddComboBox.cboState.Value
      End If
      frmAddComboBox.Hide
End Sub
```