

IFA/QFN VBA Tutorial

Notes prepared by Keith Wong

Chapter 4: Events and Error Handling

4-1: Errors

There are two types of errors: Compile-time error and Runtime error. Compile-time errors are those which can be detected when the program are compiling, for example, syntax errors. Runtime errors are those which can only be detected while the program is running, for example, logical errors, overflow errors.

4-1-1: Raising Runtime Errors

We can use the `Err` object (belonging to the class `ErrObject`) in VBA to raise and handle runtime errors. As shown before, we can use the `Raise` method to generate an error. The full syntax is as follows:

```
Err.Raise number, source, description, helpfile, helpcontext
```

- **number**: Required. Long integer that identifies the nature of the error. Visual Basic errors (both Visual Basic-defined and user-defined errors) are in the range 0 – 65535. The range 0 – 512 is reserved for system errors; the range 513 – 65535 is available for user-defined errors. When setting the `Number` property to your own error code in a class module, you add your error code number to the `vbObjectError` constant. For example, to generate the error number 513, assign `vbObjectError + 513` to the `Number` property.
- **source**: Optional. String expression naming the object or application that generated the error. When setting this property for an object, use the form `project.class`. If `source` is not specified, the programmatic ID of the current Visual Basic project is used.
- **description**: Optional. String expression describing the error. If unspecified, the value in `Number` is examined. If it can be mapped to a Visual Basic run-time error code, the string that would be returned by the `Error` function is used as `Description`. If there is no Visual Basic error corresponding to `Number`, the "Application-defined or object-defined error" message is used.
- **helpfile** (Advanced): Optional. The fully qualified path to the Help file in which help on this error can be found. If unspecified, Visual Basic uses the fully qualified drive, path, and file name of the Visual Basic Help file.
- **helpcontext** (Advanced): Optional. The context ID identifying a topic within `helpfile` that provides help for the error. If omitted, the Visual Basic Help file context ID for the error corresponding to the `Number` property is used, if it exists.

You can also check the values of `number`, `source` and `description` after an error being raised. You can use the corresponding properties to do so.

- `Number` Property
- `Source` Property
- `Description` Property

For example,

```
If Err.Number = 11 Then
    ' Do Something
End If
```

The default property of Err is Number. We can omit .Number as you call the property. So the following code is the same as above.

```
If Err = 11 Then
    ' Do Something
End If
```

4-1-2: Trapping Errors

VBA has some runtime error handling abilities so as to help you get rid of the Macro Error dialog. You may find essential to use these abilities in certain kinds of programming. You can trap runtime errors by On Error statements, which has three forms.

- On Error Resume Next
- On Error GoTo *label*
- On Error GoTo 0

4-1-2-1: On Error Resume Next

With On Error Resume Next, the program will simply continue with the next instruction. (That means even there is an error occurring, no Macro Error box will appear.) This can be very helpful in some cases, especially in file or database handling.

Example:

```
Sub demo1(arg1 As Variant)
    'arg1 maybe an array of integers or an integer

    Dim someValue As Integer

    On Error Resume Next
    someValue = arg1
    someValue = arg1(0)

    MsgBox someValue
End Sub
```

4-1-2-2: On Error GoTo label

With On Error Goto *label*, the program will go to the part in the current procedure with a line label marked *label*. You can set up an error handling routine there. You can specify a line label at any line in the procedure, with a name following the naming constraints of a variable, and ending with a colon (:).

At the end of the error handling routine, you can specify the method of continuation of your program. There are three kinds of Resume statements which can help you do the job:

- Resume
re-executes the instruction on which the error occurred.
- Resume Next
resumes the procedure with the next instruction
- Resume *label*
resumes the procedure at *label*

4-1-2-3: On Error GoTo 0

This simply means VBA uses the “normal” behavior of error handling: using a Macro Error dialog box when error occurs. Usually you use it to deactivate previously inserted On Error statements.

Example:

```
Sub ch2_11_1_ex()  
  
    Dim n As Integer, i As Integer, sum As Double, numInput As Double  
  
    On Error GoTo ErrorHandler  
    n = InputBox("Enter n")  
    On Error GoTo 0  
  
    sum = 0  
  
    For i = 1 To n  
        On Error GoTo ErrorHandler  
        numInput = InputBox("Enter number " + CStr(i))  
        On Error GoTo 0  
        sum = sum + numInput  
    Next i  
  
    MsgBox "The sum is " + CStr(sum)  
    On Error GoTo ErrorHandler2  
    MsgBox "The mean is " + CStr(Round(sum / n, 2))  
  
Exit Sub  
  
ErrorHandler:  
    If Err = 13 Then ' Type mismatch. Notice the default property for Err is number.  
        ' We can omit .number here.  
        MsgBox "Your input is incorrect. Please input again."  
    Else  
        Err.Raise Err ' If not type mismatch, raise the error so that Macro Error box appears.  
    End If  
    Resume  
  
ErrorHandler2:  
    If Err = 6 Or Err = 11 Then ' Overflow or division by zero  
        MsgBox "The mean is 0"  
    Else  
        Err.Raise Err  
    End If  
    Resume Next  
  
End Sub
```

It seems that you cannot handle any errors which occur in an error handling routine by On Error statements.

Let's try! Corner

Edit the first exercise you have done in 3-10 (available in Ch3.bas, Curve.cls, Line.cls, QuadraticCurve.cls). Instead of inputting the four specified lines or quadratic curves, let the user input the parameters of the curve of their choice.

- 1 Ask the user to input the number of lines he wishes to input.
- 2 Ask the user to input the lines (with a point and a slope).
- 3 Ask the user to input the number of quadratic curves to input.
- 4 Ask the user to input the quadratic curves.
- 5 Modify your program to reflect the changes
- 6 Design the user input error handling routines by referring to the previous example on how to handle number inputs.
- 7 Test your program.

4-1-3: Error Handling in Nested Procedures

Suppose that procedure A is the main program, A calls B, and B calls C. If an error occurs in C, the error handling routine belonging to C will be called. If there is no error handling routine in C, then control is returned to B. If there is no error handling routine in B, the control is passed back to A. Only if there is no error handling routine in A does the Macro Error dialog appear.

4-2: Events

Events are some significant actions happened when the program is running. Encountering error is an example of an event. In fact, we have seen a few more examples of events before. When an object is created or destroyed, there is an event. VBA captures the event and invoke `Class_Initialize` or `Class_Terminate`, if there is one in a class, so that some user codes can be run.

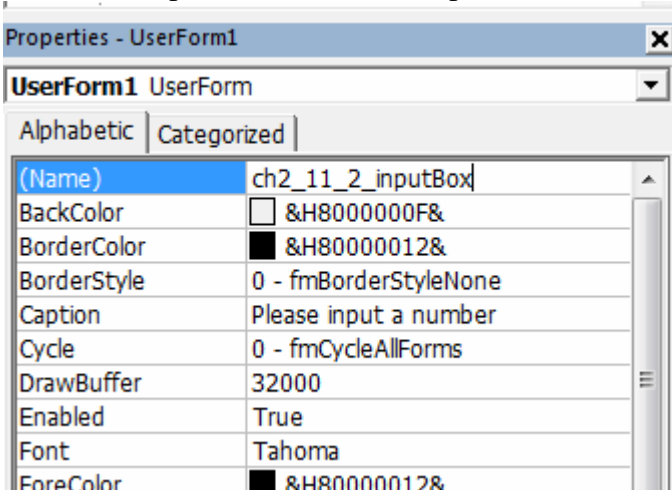
In order to familiarize with events, let's have a quick introduction on how user defined forms, which depends heavily on events, can be created and used in VBA.

4-2-1: Creating a Form

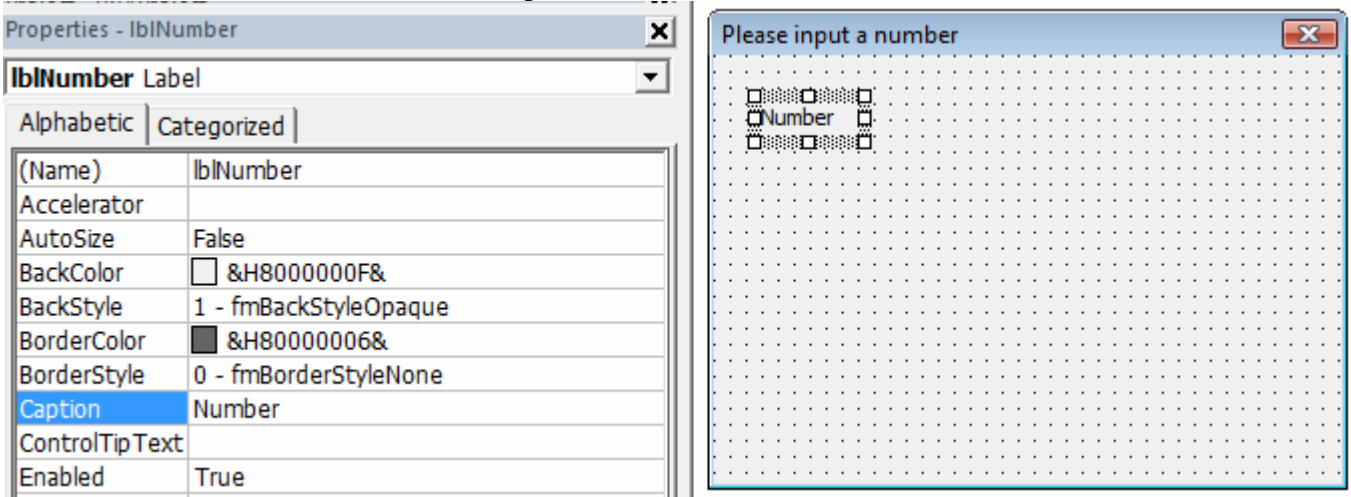
Let's try! Corner

In 2-11-2, we have used several input boxes to let the user input numbers until the Cancel button is clicked. Then the sum and mean of the inputted numbers are displayed. However, if you are smart enough you may recognize that it is not a very good user input error control method. The reason is that the input box returns an empty string when the user clicks Cancel, but if the user does not input anything and click OK, the result is also an empty string. The program cannot distinguish whether the user mistakenly clicked OK or intentionally clicked Cancel. This motivates us to design a custom input box to handle the problem.

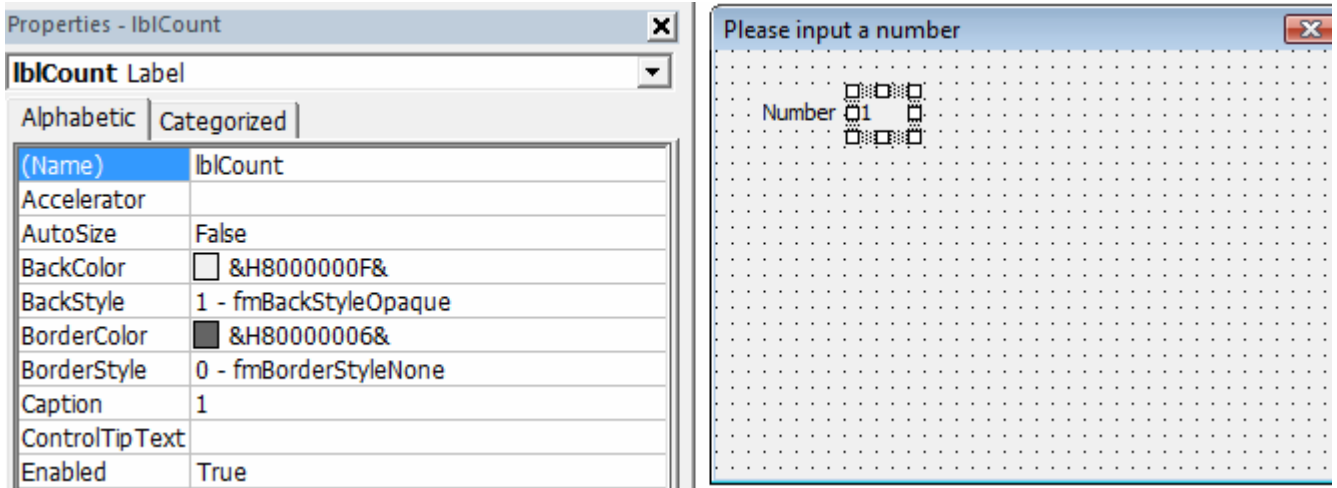
- 1 Insert a user form from Insert > User Form
- 2 In the Properties pane, change the name and the caption (title) of the form. Let the name be ch2_11_2_inputBox, and let the caption be "Please input a number".



- 3 In the main window, notice a blank form. Click once on the blank form to bring it in focus. Notice a toolbox appears.
- 4 Notice three controls we are going to use: the label control (Row 1 Column 2), the textbox control (Row 1 Column 3), and the command button control (Row 3 Column 2).
- 5 Add a label "Number" with name and caption shown as below:



- 6 Add a label “1”:



- 7 Add a textbox named “txtNumberInput”, a command button captioned “Add” and named “btnAdd”, a commandbutton captioned “Finish Input” and named “btnFinish”.



4-2-2: Programming the event triggers

After the user interface is designed, we can add some program codes in the event triggers.

Let's try! Corner

We want the computer to respond when either button is pressed. Therefore we want to capture the events of clicking buttons.

- 1 Double click the button “Add”. The window has changed to code mode so that you can add your own code. Notice the procedure name “btnAdd_Click”. The name tells you that it is an event trigger when the button “Add” is clicked.
- 2 We want to check if the number in the textbox is actually a number. If so, temporarily hide the input box so that the main procedure can get access to the controls in this input box. Remember forms and controls are objects, so you can access their properties and methods by the period (.) operator. Also, the values of all the controls have been set to default, so that when you access the value properties, you can just write the object name. The value of a textbox can be accessed with the Text property. You can omit .Text when you access this property. Sample code:

```
Private Sub btnAdd_Click()
    If IsNumeric(txtNumberInput) Then 'check if the text in the textbox is a number
        Hide 'hide the input box and return the control to the procedure that calls it
    Else
        MsgBox "You should input a number!", vbExclamation, "Error input"
        With txtNumberInput
            .SelStart = 0
            .SelLength = Len(.Value)
            .SetFocus 'Set the focus and selection of the textbox
        End With
    End If
End Sub
```

- 3 In the same way, double click the button “Finish Input” and input the code to hide the input box. We also want to notify the main program that the finish button is clicked. We set the textbox input to “Finish”.

```
Private Sub btnFinish_Click()  
    txtNumberInput = "Finish"  
    Hide  
End Sub
```

- 4 At the top of the code window, click UserForm at the left and then QueryClose at the right.
5 Add the following code. This is to deal with the “X” button on the upper right corner of the input box being pressed. In such case, simply clear the content in the textbox, so that the main program can distinguish this case from the above two.

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)  
    If CloseMode = vbFormControlMenu Then txtNumberInput = ""  
End Sub
```

For more information on how to use controls, read the online manual (msdn.microsoft.com) or read the suggested textbooks.

4-2-3: Showing and Getting Values from a Form

Let's try! Corner

- 1 We start changing the procedure ch2_11_2_ex (in ch2.bas). First, change the type of numInput to Double.
2 Next, change the codes in Do loop. Instead of outputting the default InputBox, we show our custom box (the ch2_11_2_inputBox object) by invoking the Show method. After the input box is shown and inputted, we get back the textbox control with ch2_11_2_inputBox.txtNumberInput. Do some simple manipulation with this control object.
Also, remember to update the value of the Count label before showing the inputBox, and remember to unload the form after the Do loop.

Sample Code (the Do loop):

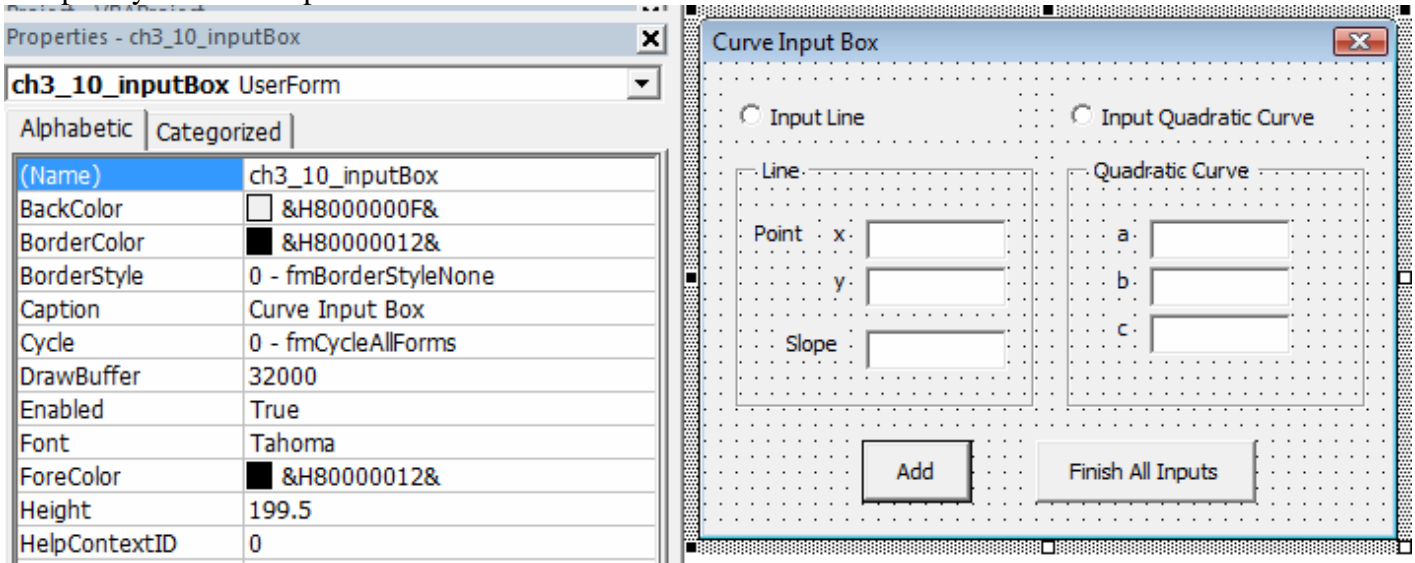
```
Do  
    n = n + 1  
  
    With ch2_11_2_inputBox  
        .lblCount = n 'Update the count label  
        With .txtNumberInput 'Update the selection and focus  
            .SelStart = 0  
            .SelLength = Len(.Value)  
            .SetFocus  
        End With  
  
        .Show 'Show the input box  
  
        If IsNumeric(.txtNumberInput) Then 'If Add button is clicked  
            numInput = .txtNumberInput  
        ElseIf .txtNumberInput = "Finish" Then  
            n = n - 1  
            Exit Do  
        Else  
            Exit Sub  
        End If  
    End With  
  
    sum = sum + numInput  
Loop  
  
Unload ch2_11_2_inputBox
```

- 3 The procedure is ready to use and you can test your new input box.

Let's try! Corner

Modify your exercise in 4-1-2-3. Create an input form for user to input conveniently. You may want to use the Radio Button (Option Button) control (Row 2 Column 3). The default property for a radio button is Value. It is True if selected and False if not selected.

Sample layout of the input form:



Exercise 4

1. Construct your own error routines and user forms for Question 6 in Exercise 2.
2. Construct your own error routines and user forms for Question 2 in Exercise 3.
3. Construct your own error routines and user forms for the Stock example in 3-10 (possibly also Question 4 in Exercise 3).