

# VBA Foundations, Part 8

A Tutorial in VBA for Beginners—The Eighth of a Twelve Part Series

Richard L. Binning / [rbinning@attbi.com](mailto:rbinning@attbi.com)

If only Error messages were easier to understand. The sad truth is that usually, by the time you are shown the error message, it is frequently much too late to save your project. Don't worry too much about it though; you will get another opportunity to type it all back in to the editor, and another, and another. But soon, the error messages will become less and less frequent, until they are but a distant memory. Then, at long last your code might be ready for others to try. And then the circle begins anew; only this time you don't see the message. This time you get a phone call from someone who:

- Probably won't be able to adequately describe the message they are seeing.
- Is not sure how far your routine got before the error message appeared.
- Is confident that *they* did nothing wrong it *must* be bad programming!

Perhaps Thomas Paine said it best, "These are the times that try men's souls." "Too strong", you say? "Errors? Not me!" you say? You may rest assured that it **will** happen and usually sooner than later, but you should take comfort in this issue's lessons. In this lesson, we will learn a little about errors, *what* types of errors can occur, *why* they occur, and *how* to apply some preventive medicine so that they are manageable *when* they occur. Because occur they will, usually at the most inopportune of moments, during demonstrations or training sessions. This is the undeniable truth of programming. No matter how thoroughly you test your program, errors can still crash it during normal operation. So before you get carried away pressing that "shiny black right-facing triangle" in the middle of the visual basic editor toolbar, read this article to the end and hopefully you'll take a way a gem or two to guide you.

Scared? Don't be! It's not really that hard, if the truth be told. So, since you and your users won't be presented with a pithy Error Haiku message, perhaps we should just get started.

This brings us to the first Law of Error Handling, namely *Use Option Explicit*. I mentioned this Law in a previous article, but it is so vitally important as you begin to write your routines that I must repeat it here. Use Option Explicit for the following benefits:

- Make sure there are no misspelled variables in your code.
- Make sure there are no variables being used which have not been declared.
- Make sure that the data you are storing in the variable is of the right "Type" such as string, integer, variant, etc.

*The first time you attempt to run your code the editor will skim through your code and identify each variable that is being used and look for the "Type" of variable it is. It checks this by looking for the line where you declared the variable. If it cannot find the declaration (Dim, Public, Private, etc.), then it will halt the routine and highlight the variable where it is first used, so that you may properly declare it at the beginning of the highlighted procedure. This is usually the first hurdle, and the one that could keep you up at night trying to figure out why your code, so well thought out, simply does not perform as intended.*

Errors come in many forms and there is an entire family of error handling statements to help ensure that you produce "Good Code" (*see previous articles for definition*). In order to begin the process of writing error handlers, it will help to memorize the following statement: "On Error", and repeat it with me, "On Error". On Error, two words with more innate ability were probably never typed. Ok, I am being a bit over the top aren't I? Perhaps "never typed in the VBA editor window" is a more fitting end to that sentence. But these two words are

# VBA Foundations, Part 8

A Tutorial in VBA for Beginners—The Eighth of a Twelve Part Series

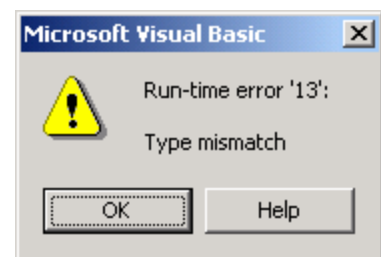
Richard L. Binning / rbinning@attbi.com

very powerful and should begin every error handling snippet or function you will probably ever write, so memorizing them will be a good thing. Let's explore the use of the different flavors of the On Error statement.

**On Error Resume Next** is probably the simplest of the Error Handling family and is considered an in-line method of error handling. This handy phrase is completely self contained and self-descriptive. When this phrase is placed in your code, any errors encountered after that phrase will simply be ignored and the program will proceed to the next available line of code. How can we put this to use? Consider our ever popular routine Close\_N\_Purgeall.

```
Public Sub Close_N_PurgeAll()  
    On Error Resume Next  
    Dim objDrawing As AcadDocument  
    Dim objDwgs As AcadDocuments  
    Set objDwgs = Application.Documents  
        'For each drawing that is currently open in AutoCAD  
    For Each objDrawing In objDwgs  
        'I would like to automatically switch to each drawing  
        objDrawing.Activate  
        'purge the drawing  
        objDrawing.PurgeAll  
        'save the drawing  
        objDrawing.Save  
        'close the drawing  
        objDrawing.Close  
        'continue doing these steps until  
    Next objDrawing  
    'no drawings are open  
    'then close AutoCAD  
End Sub
```

We can see that the On Error Resume Next is in place, but the routine is currently working so how do we know what it is doing for us? Open up your trusty VBA Manager dialog and load up the Purgeall project. Find the “On Error Resume Next” line in your code and let's change it to see just what it is doing for us. Place a single apostrophe at the beginning of line (hint: it should turn green in your VBA editor!). Now open up four or so random drawings and let's run this code. Works just fine right? That's because our code is fairly simple and already debugged. Now let's change some of the code so that we experience an error by finding the line that reads like this: “Dim objDwgs As AcadDocuments” move your cursor to the end of the line and hit a backspace. This will remove the “s” from the declaration effectively changing the type of the object from a collection of drawings to a single drawing document or file. Now run the routine. You should see the following message box appear (see graphic right.). The reason this message appears is because you are trying to populate a single document variable with a collection of documents. This brings us to a key point. If you are working with code and you realize that it doesn't seem to be working as intended, check for any “On Error” statements and try commenting them out and running your code. Back to the example, so how does the On Error Resume Next actually work? The code is convinced that the error is not an earth shattering event, and is coaxed to pick up the pieces and continue performing. We can see this in action by “uncommenting” the “On Error Resume Next” line



# VBA Foundations, Part 8

A Tutorial in VBA for Beginners—The Eighth of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

by finding the single apostrophe at the beginning and deleting it. Now open up four or so random files and re-run the code with the error handler in place. What happens? No drawings are cycled, and no drawings are saved or purged. But this occurs without an error being registered. You can verify by commenting out the handler again if you wish. Conclusion: This is a useful error handling method, but it doesn't report when or why it occurs. Remember this because we will make extensive use of this method when working with objects later in this series. We should use this method when: we are not concerned with why it occurred; or when we expect an error. More on the use of this method (by exception) later.

Before I get too deep in this handy family of phrases, let me point out the fact that every sub, function, or procedure you write should ideally contain some error handling. "Why", you ask? "Because", the On Error statement **only** turns on error trapping for the procedure (Sub or Function) in which the phrase is found or until a new statement is encountered. Therefore, you must use a type of On Error statement in every Sub or Function in which you want error trapping to occur. You won't go wrong if you remember this, the Second Law of Error Handling.

The rest of the family involves a more direct re-direction. (oxymoron or pun? You decide). Although, hard-core programmers may try to convince you that a truly well written program, or language for that matter, would never have any GoTo statements in it, I disagree. One of the most compelling reasons people have adopted Visual Basic in such great numbers is it's relative ease of use and the fact that it is mostly driven by reacting to events such as button clicks, mouse movements, and the like. This type of environment is ideally suited to the use of GoTo statements, although most VB programmers prefer to limit their use of GoTo statements to their Error Handling snippets. This I **can** agree with. Let me emphasize that the use of GoTo statements in other areas of your program can make your program **hard to debug** or **understand**, especially after it has been on the shelf a while. But please **DO** use GoTo statements in Error Handlers where they are **almost always appropriate** and usually the cleanest solution to the problem at hand.

So, on with the show, *On Error GoTo SomeOtherMeaningfulLabel* is used when you expect an error and wish to deal with that error prior to resuming your code. This is an important aspect of it's use: You must eventually include some form of the keyword "**Resume**" at a point after your program has jumped to the "**SomeOtherMeaningfulLabel:**" line or you should clean up its objects and exit the sub or function it is contained in. Therefore you will usually use a GoTo keyword with one or more Labels. The first label accepts the redirected execution and attempts to handle the error, and is usually the label immediately following the keyword. The second label can be used as an alternate avenue of completion to complete your code after the error condition is fixed, provided that your intention isn't to exit the sub on the first error encountered.

Lets allow that to sink in while we add some error handling to our routine. We will unload the Purgeall routines without saving and then reload them into the VBA editor now. The items we will use in this exploration include our standard close-n-purgeall routine, two labels, and a little luck. Take a look at the following code and add the modified lines as shown in RED and don't forget the liberal comments included as shown below:

```
Public Sub Close_N_PurgeAll()  
On Error GoTo Error_Catch '(Label 1=Error_Catch)Lets redirect our Error _  
    when it occurs so we can deal with it  
Dim objDrawing As AcadDocument  
Dim objDwgs As AcadDocument 'Note the error here, we will fix this later!  
Set objDwgs = Application.Documents
```

'For each drawing that is currently open in AutoCAD

# VBA Foundations, Part 8

A Tutorial in VBA for Beginners—The Eighth of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

```
For Each objDrawing In objDwgs
    'I would like to automatically switch to each drawing
    objDrawing.Activate
    'purge the drawing
    objDrawing.PurgeAll
    'save the drawing
    objDrawing.Save
    'close the drawing
    objDrawing.Close
    'continue doing these steps until
Next objDrawing
'So far so good, no errors we couldn't handle
'Lets clean up our objects and go home
'No sense in rewriting code so lets use our Error_Exit
GoTo Error_Exit
```

'NOTE: We will come back to these next two lines in a future article  
'no drawings are open  
'then close AutoCAD

```
Error_Catch: 'This is Label 1 and the first Error Handling redirection
'If code gets here than we have got an error
MsgBox "Error Detected!" & vbCrLf & "Error #" & Err.Number & vbCrLf & Err.Description
Err.Clear 'Lets clear the error and try again
On Error GoTo 0 'The On Error GoTo 0 statement turns off error trapping.
On Error GoTo Error_Exit 'Turn it back on and give it a different exit point
Resume 'Note this attempts to run the same line that kicked out the error last time.
'For extra points change the above line to "Resume Next" and make note of the difference
'where could we make use of this???
```

```
Error_Exit: 'Label 2 nothing good happening so lets clear the books
'If code gets here than we may have an error we couldn't handle so lets _
clean up our objects, post a message and exit gracefully
'Clean up objects
Set objDrawing = Nothing
Set objDwgs = Nothing
'Check for an error
If Err.Number > 0 Then
    MsgBox "Error Will Robinson, Warning!, Warning!" & vbCrLf & "Error #" & Err.Number & _
vbCrLf & Err.Description
Else
    'no Error detected
End If
'closing down now!

End Sub
```

Now we'll run this code a line at a time in the editor. Please open the code editor and place the cursor at the beginning of the following line: "Public Sub Close\_N\_PurgeAll()". Now we will begin to step through the code

# VBA Foundations, Part 8

A Tutorial in VBA for Beginners—The Eighth of a Twelve Part Series

Richard L. Binning / [rbinning@attbi.com](mailto:rbinning@attbi.com)

and follow its execution line by line. Hit the “F8” key and you should see the line mentioned above is now highlighted in yellow. We are now running in debug mode a line at a time. This is a good way to execute code and verify (debug) your code and find errors or just understand it a little better. (Stay tuned to next issue for more debugging hints, tips and tricks.) Continue hitting the “F8” key and following your code through to completion. Be sure to read the comments I’ve included to help explain the direction the code is taking. Now that you have stepped through and fully understood the code and the error trapping, lets put that “s” back in and make this routine useful once more. Now your code is both useful and full of error handling.

Lets recap what we have done in this segment. Your options with the GoTo keyword include Resume;(to immediately retry the same line of code that resulted in the error – **Note:** Ensure that you’ve corrected the Error Because if you use Resume by itself, without a label, then your code will jump back directly to the same statement it came from originally), Resume Next;(to skip to the line after the line which resulted in an error), or Resume SomeLabel;(to go to an alternate path of code completion as necessary).

This topic could not be completed without a mention of the built in Err object so thoughtfully provide by the VBA language itself and used in our example. The Err object has a number of properties and methods built in and is much to in-depth to cover adequately in this article, so I leave the rest of this up to your research.

Like any aspect of programming, a good error-handling routine should be "invisible" to the user. If you have an idea of which kinds of errors to expect, you can troubleshoot potential problems without the user ever knowing something's gone wrong. On the other hand, if your error-handling routine can't figure out what to do, you can at least print out error messages in plain English or perhaps Haiku format? and give the user a chance to recover from the problem without a crash. Whatever situation you may find yourself in, error trapping is an important and effective way to make your programs polished, professional, and easier to run.

I hope you have found this exploration of VBA error control both informative and useful. As always, make sure to search through the on-line help whenever you have a question related to this information. If you are really stumped, please send in your questions to the VBA guild or the email address at the top of this article. See you on the guilds, at AU or in the next issue of AUGIWorld™ magazine coming soon to a mailbox near you.