Managing VBA Errors

Note If you are new to Microsoft Visual Basic for Applications (VBA), see Chapter 23, "VBA Primer," before continuing here.

Errors can occur when you compile a VBA project or when you attempt to run a macro. Following are a few of the most common error types, along with their corresponding error numbers.

- "Type Mismatch" (error number 13) indicates that you're trying to act on an item in an unavailable way, such as if you define an integer variable for the response to an input box that requires a text string response. Note that the absence of this error doesn't guarantee that you've assigned the correct data type to your variable because VBA can automatically allow for incorrect data types under some circumstances.
- "Method or Data Member Not Found" (error number 461) usually indicates that you either misspelled a term or you referenced an item that doesn't exist.
- "Requested Member of the Collection Does Not Exist" (error number 5941) appears when you run a macro on a range that doesn't include the object you specified to act on. For example, this would occur if your macro includes the statement `ActiveDocument.Tables(3).Delete` and two or fewer tables are in the active document.
- "Object Required" (error 424) indicates that you need to specify an object upon which to act. This error might occur, for example, in a `For Each…Next` loop when you don't correctly define the collection of objects upon which you want to act. A similar error may appear as "Object Doesn't Support This Property or Method."

Note Error numbers are provided in the preceding list because it is possible to correct for individual errors—that is, to write code that enables your macro to respond differently to different errors. When you do this, you need to identify the error by number, as discussed later in this article, under the heading "Creating Error Handlers."

When an error occurs while you're compiling a project, the error statement is selected and a message box appears telling you the type of error and offering help. If you click the help option in Word 2010, an article explaining the error type is opened in most cases.

For Mac Users

As discussed in Chapter 23, VBA help in Office 2011 refers you to an online resource that doesn't always have the information you might need. Instead, you can get much more complete help from the applicable program developer home pages for the Office 2010 versions of Word, PowerPoint, and Excel—most of which applies to Office 2011 VBA as well.

Word VBA Help and developer reference:
http://office.microsoft.com/client/helphome14.aspx?lcid=1033&NS=WINWORD%2EDEV&Version=14

PowerPoint VBA help and developer reference:
http://office.microsoft.com/client/helphome14.aspx?lcid=1033&NS=POWERPNT%2EDEV&Version=14

Excel VBA help and developer reference:
http://office.microsoft.com/client/helphome14.aspx?lcid=1033&NS=EXCEL%2EDEV&Version=14

If an error occurs when you run a macro, a similar message box appears to indicate the error type, but it provides the options End, Debug, and Help. Just as with compile errors, the help option usually provides an article with more information on the error type. (Note also that clicking the help option doesn't close the error message for either compile or runtime errors.) Click End to dismiss a runtime error message if you're familiar with the error and know where to find it. Or, click Debug for guidance in correcting a runtime error.

When you click Debug, the statement that VBA sees as having caused the error is highlighted. (Note that, by default, the statement is highlighted in yellow with a yellow arrow at the margin indicator bar, but you can customize this formatting in the Options dialog box (Preferences in Office 2011).) This also puts the program into Break Mode, which stops code from executing. When this occurs, "[break]" appears in the title bar after

the project name. To exit Break Mode, which removes the highlight from the error statement, click the Reset button on either the Standard toolbar or the Run menu. (Note that Break Mode also restricts many actions in the application window. So, resolve the error and turn off Break Mode before you return to working in the applicable program.)

If a compile error occurs at runtime (meaning that you didn't compile the code, but ran a macro that contains a compile error), the project goes into Break Mode when you click OK to dismiss the error message. The title line of the macro that caused the error is highlighted and the error statement is selected.

Testing Macros and Debugging Errors

When you're not sure of the cause of an error, there are several ways to go about finding it efficiently. Some of these options can also be used when you want to see the results of just a portion of your code at a time, even if errors don't occur. A few of the easiest options for testing or debugging code are described in the following list:

- You can step into a macro to execute one line at a time and see the results as you go. To do this, click in the macro and then press F8 (Shift+Command+I in Office 2011) to start executing the macro one line at a time. Or, on the Debug menu, click Step Into. Use the same keyboard shortcut or menu option again each time you want to run the next statement.

  You can also click in a specific line in the macro and then press Ctrl+F8 (Command+\ in Office 2011) to execute the macro up to the line where your insertion point appears.

- Use a breakpoint to run a macro using a traditional method (such as by pressing F5 from the Visual Basic Editor or running the macro from the Macros dialog box on the Developer tab), but execute statements only up to a specified line of code. To add a breakpoint, click the margin indicator bar to the left of the line of code. A circle appears on the window edge where you clicked, and the statement is shaded (in burgundy, by default), as you see in the following image.

If you're not sure where to click, you can right-click in the statement instead, point to Toggle, and then click Breakpoint. To remove a breakpoint, just click the circle that

appears to the left of the statement. (As with error notifications, you can customize the formatting of breakpoints in the Options (Preferences) dialog box.)

- You can comment out a block of code, so that it gets skipped when a macro runs. This is a great tool to use when you want to try running just part of your macro, especially if you have lines of code that aren't finished (which would otherwise throw an error when you run the macro). To do this, on the View menu, point to Toolbars, and then click Edit to open the Edit toolbar. On that toolbar, you'll find Comment Block and Uncomment Block commands shown below, which add or remove an apostrophe from in front of each selected line of code.

Office 2010 Office 2011

For Mac Users

If you've tried to customize VBA settings in the Preferences dialog box and they've not held (as mentioned in Chapter 23), note that this issue is fixed in Office 2011 Service Pack 1. This service pack also fixes the issue that prevented Word 2011 from automatically loading previously-loaded global templates (VBA add-ins) on subsequent Word sessions.

Creating Error Handlers

When you write macros, particularly for others to use, it's a good idea to try to account for any errors you can control. You can account for many possible errors with conditional structures to protect the macro from conditions you know would cause an error. For example, if the macro acts on the active document, you can set the macro to end without taking any action if no documents are open.

However, you might also want to set up error handlers to manage what the user sees if an unexpected error occurs. In some cases, you might even know that certain conditions could cause an error, but find it more efficient to add an error handler than to write code to account for every possibility.

The two most common types of error handlers are the statements `On Error Resume Next` and `On Error GoTo ErrorHandler`. Let's look at those one at a time.

Use `On Error Resume Next` when a given instance of a loop, for example, might throw an error under certain conditions, but you'd still want the loop to continue running after it encounters an error. To do this, type `On Error Resume Next` on its own line early in the macro, before any code that could possibly throw the error for which you want the handler to correct. Then, if the possible error occurs, the user won't be notified—the code will just skip the instance it couldn't act on and continue on its merry way.

Use `On Error GoTo...` when you want to control what happens when an error occurs. For example, you might write an error handler that contains a message box telling the user that an error has occurred and what to do to correct it. Take a look at the following example:

```
Sub Sample()
On Error GoTo MyHandler
<code statements>
Exit Sub
MyHandler:
MsgBox "Please place your insertion point in the table you want to copy before
running this macro.", vbExclamation, "Please Try Again"
End Sub
```

Notice that this handler consists of an `On Error GoTo...` statement as well as another statement with the name of the handler followed by a colon (both in bold in the preceding sample). You can name the handler anything you like, within VBA naming conventions.

> Note Notice the `Exit Sub` statement that precedes the error handler, so that a macro that doesn't throw an error stops executing code before the error handler is executed. `Exit Sub` is usually better to use than the `End` statement, because it only affects the active procedure. The `End` statement ends all code execution. `Exit Sub` exits the active macro but continues running any other code, if the active macro was called from another procedure.

What this error handler indicates is that, if an error occurs, the code stops executing and moves to the line following `MyHandler:` to continue executing code from that point. In this case, the handler just displays a message box giving the user information on why they couldn't run the macro.

> **Note** A term followed by a colon is known as a Line Label. It's simply a way of naming a position in your code. If you weren't using an error handler, you could just type `GoTo SampleName` where you want the code to pick up at another position in the macro and then precede the new position with a line that reads `SampleName:`.

> **Caution**! If you add an error handler of any kind to your macro, be sure to comment that line out (add an apostrophe in front of the statement) when you test the macro. Otherwise, any errors might be ignored, causing you not to see the reason for an unexpected result. Also remember that an error handler can't account for code that comes before it. So, be sure to place an error handler before any code that might cause the error you want to address.

When deciding on how to handle errors for a particular macro, consider whether accounting for possible conditions or adding an error handler is the more effective way to go. For example, in the preceding code sample, if you just need the user to click in a table before running the macro, you might have added the following code at the beginning of the macro instead of an error handler.

```
If Selection.Information(wdWithinTable) = False Then
MsgBox "Please place your insertion point in the table you want to copy  before
running this macro.", vbExclamation, "Please Try Again"
End If
```

Trapping Individual Errors

As mentioned earlier, you can identify different actions to take in an error handler based on the type of error. One way to do this is to use a `Select Case` conditional structure.

> Note To learn about Select Case and other conditional structures, see Chapter 23.

Similar to an `If...Else...End If` structure, the `Select Case...End Select` structure enables you to specify different actions based on conditions you identify. Select Case is not at all strictly for error handlers, but trapping individual errors offers a good example of using this construct.

While `If...End If` structures evaluate each `If`, `Else If`, or `Else` expression independently, you can use a `Select Case` structure when you want to compare several possible results to a single expression. Take a look at the following code, for example, that uses one of the input box macro examples from Chapter 23.

```
Dim myInp As Integer
ResumeInputBox:
On Error GoTo ErrorHandler
myInp = InputBox("How many columns would you like?", "My Input Box")
With Selection
   .Tables.Add Range:=.Range, NumRows:=5, NumColumns:=myInp
End With
End
```

```
ErrorHandler:
Select Case Err.Number
   Case 13
         MsgBox "Please enter a numeric value to continue.", vbInformation
         Resume ResumeInputBox
   Case Else
         Msgbox Err.Description
End Select
```

Similar to an `If…End If` structure, you can identify several cases with the `Select Case` structure and provide for all cases not specified with a `Case Else` statement. As with all paired structures, remember to add `End Select` at the end of the structure, or your code will return an error.

Getting Help with Errors

In Office 2010 VBA, you can search for help using the Type A Question For Help box on the Visual Basic Editor menu bar to get specific help. But, in both Office 2010 and Office 2011, there are often faster ways to get to exactly what you need when searching for help with errors.

- In Office 2010, the Help command in error message boxes takes you directly to a help article on that specific error message.
- In Office 2010 and Office 2011, you can use the VBA help article Trappable Errors: http://office.microsoft.com/client/helppreview14.aspx?AssetId=HV080154317&lcid=1033&NS=POWERPNT%2EDEV&Version=14&respos=0&CTT=1&queryid=847db575701745448b81e5ec72ebe013.
    - This article is also available in Office 2010 from the Visual Basic Editor Help in Word, Excel, and PowerPoint. You can then use Ctrl+F for the find feature, to quickly locate the name or number of the particular error you need. The Trappable Errors article lists each error with a hyperlink to its article.
    - For Office 2011 users, you can use the links provided earlier to online VBA help and then use the search box on those developer home pages to search for a VBA help topics as well—which is exactly what I did to provide the direct link to the Trappable Errors article provided here.