

VBA - ERROR HANDLING

http://www.tutorialspoint.com/vba/vba_error_handling.htm

Copyright © tutorialspoint.com

There are three types of errors in programming: *a* Syntax Errors and *b* Runtime Errors *c* Logical Errors.

Syntax errors

Syntax errors, also called parsing errors, occur at interpretation time for VBScript. For example, the following line causes a syntax error because it is missing a closing parenthesis:

```
Function ErrorHanlding_Demo()  
dim x,y  
x = "Tutorialspoint"  
y = Ucase(x  
End Function
```

Runtime errors

Runtime errors, also called exceptions, occur during execution, after interpretation.

For example, the following line causes a runtime error because here syntax is correct but at runtime it is trying to call `fnmultiply`, which is a non-existing function:

```
Function ErrorHanlding_Demo1()  
    Dim x,y  
    x = 10  
    y = 20  
    z = fnadd(x,y)  
    a = fnmultiply(x,y)  
End Function  
  
Function fnadd(x,y)  
    fnadd = x+y  
End Function
```

Logical errors

Logic errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.

You can not catch those errors, because it depends on your business requirement what type of logic you want to put in your program.

For example, dividing a number by zero or a script that is written which enters into infinite loop.

Err Object

Assume if we have a runtime error, then the execution stops by displaying the error message. As a developer, if we want to capture the error, then **Error** Object is used.

Example

In the below example, **Err.Number** gives the error number and **Err.Description** gives error description.

```
Err.Raise 6      ' Raise an overflow error.  
MsgBox "Error # " & CStr(Err.Number) & " " & Err.Description  
Err.Clear       ' Clear the error.
```

Error Handling

VBA Enables an error-handling routine and can also be used to disable an error-handling routine. Without an On Error statement, any run-time error that occurs is fatal: an error message is displayed, and execution stops abruptly.

```
On Error { GoTo [ line | 0 | -1 ] | Resume Next }
```

Keyword	Description
GoTo line	Enables the error-handling routine that starts at the line specified in the required line argument. The specified line must be in the same procedure as the On Error statement, or a compile-time error will occur.
GoTo 0	Disables enabled error handler in the current procedure and resets it to Nothing.
GoTo -1	Disables enabled exception in the current procedure and resets it to Nothing.
Resume Next	Specifies that when a run-time error occurs, control goes to the statement immediately following the statement where the error occurred, and execution continues from that point

EXAMPLE

```
Public Sub OnErrorDemo()  
    On Error GoTo ErrorHandler    ' Enable error-handling routine.  
    Dim x, y, z As Integer  
    x = 50  
    y = 0  
    z = x / y    ' Divide by ZERO Error Raises  
  
    ErrorHandler:    ' Error-handling routine.  
    Select Case Err.Number    ' Evaluate error number.  
        Case 10    ' Divide by zero error  
            MsgBox ("You attempted to divide by zero!")  
        Case Else  
            MsgBox "UNKNOWN ERROR - Error# " & Err.Number & " : " & Err.Description  
    End Select  
    Resume Next  
End Sub
```

Loading [Mathjax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js