EXCEL – VBA – PREVENT USERS CTRL BREAK (ING) YOUR CODE DURING EXECUTION

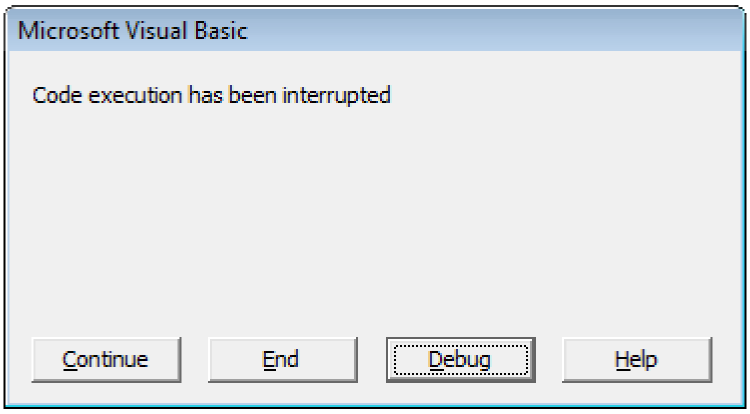
Anytime I write **VBA** code that runs for more than a split second, one of my worries remains that someone will **ctrl + break** it. You see, I am a very strong supporter of P.E.T.A. (*People for Ethical Treatment of Algorithms*) and believe that any code, no matter how long it takes (***or in my case how badly written it is***), must be allowed the dignity to complete.

And for those who believe in killing poor little VBA code(s) with a ctrl + break, I just got a neat little trick up my sleeve.

Here's how it goes:

Take for example some **VBA** code that runs ***for a few seconds***. It is important that the user let it run for that duration without stopping code execution since there are a lot of intermediate sheets, rows and columns that the code generates and subsequently deletes before exiting.

If the user stops the code execution in between, they are left with a pretty ugly spreadsheet. (now I know that opening the workbook again is always an option but hey that wouldn't be half the fun would it).


Microsoft Visual Basic

Code execution has been interrupted

So the trick to prevent **VBA** code execution by pressing **ctrl + break** is to insert this magic statement in the **VBA** code:

```
Application.EnableCancelKey = xlErrorHandler
```

The statement instructs Excel to not show the "Code execution has been interrupted" message and provides a way for the developer to tap into the **ctrl + break** action by the user. Essentially there can be three values for **Application.EnableCancelKey : xlDisabled**, **xlErrorHandler** and **xlInterrupt**.

By setting ***Application.EnableCancelKey = xlDisabled***, we are essentially telling the application to stop responding to the **ctrl + break** command from the user. If the code runs haywire … too bad.

The ***xlInterrupt*** is the normal course of action where the user can terminate the code and is the value that the application resets to after the code has run its course.

The ***xlErrorHandler*** is the one that lets the developer instruct the application generate an error (**code 18**) and then to tap into that error by using **error handling**.

Here is a code that is supposed to run for 5 seconds. If the user tries to stop the code prematurely, the **xlErrorHandler** kicks in and let the application raise an error.

This error is then tapped by the error handler (**On Error GoTo MyErrorHandler**) and error handing code, after checking for the exact error code (***<u>error code 18 in this case</u>***), lets the code execution resume from where it left off.

```
Sub code_that_runs_5_seconds()
  On Error GoTo MyErrorHandler:
  t = Timer
  Application.EnableCancelKey = xlErrorHandler
  Do While Timer - t < 5
  Loop
  MyErrorHandler:
  If Err.Number = 18 Then
   MsgBox "Stop hitting ctrl + break !!!"
  Resume
  Else
  'Do something to make your impatient user happy
  End If
  End Sub
```

Another interesting thing to note is that you can have more than one **Application.EnableCancelKey** instructions in a piece of code.

For the portions of the code over which you (***<u>the developer</u>***) want to exert control, you can have it set to **xlErrorHandler** and for the other pieces you can let the user retain it by setting it to **xlInterrupt** later down the line.

```
Sub another_code_that_runs_5_seconds()
On Error GoTo MyErrorHandler:
t = Timer
Application.EnableCancelKey = xlErrorHandler

Do While Timer - t < 5
Loop
MsgBox 1
Application.EnableCancelKey = xlInterrupt

Do While Timer - t < 10
Loop
```

```
MyErrorHandler:
If Err.Number = 18 Then
    MsgBox "Stop hitting ctrl + break"
    Resume
Else
    'Do something to make your impatient user happy
End If
End Sub
```

Go ahead – take control.

You can download an example **here**.