User Friendly Error Traps

by Clark Anderson

The day you can consistently write PERFECT applications is the day no one can afford your services!

I have seen various ways programmers have dealt with or not dealt with errors. I have also seen “users / clients / customers” become accustomed to avoiding certain uses of an application because it gives some confusing message and kicks them out of the application.

I add a simple but effective “Error Trap” to all but the simplest functions or subroutines. In the examples show here, the error trap is started with the “On Error GoTo... “ command. The “On Error” command must precede any code that you want to protect with the error trap. I use the command to direct program flow to the “error handling section at the end of the function or subroutine. I have standardized on using labels that begin with “Err_” followed by the procedure name. I have also standardized providing another label “Exit_” followed by the procedure name.

```
Sub pb_done_Click ()

On Error GoTo Err_pb_done_Click

      'procedure code here

Exit_pb_done_Click:

    Exit Sub

Err_pb_done_Click:

 Select Case Err

    Case Else: MsgBox " pb_done_Click ERROR " & Err & ": " & Error(Err) & "
occurred."

End Select

    Resume Next

End Sub
```

The next part of the “Error Trap” is the optional Exit_ label. A code line label must be followed by a colon(:). This Exit_label is followed by any code that you want to execute before leaving the procedure. Examples are : resetting the pointer or setting function value. This is followed by a VERY IMPORTANT Exit Sub or Exit Function statement to avoid processing your error trap code instead of exiting. This might get loopy.

Now you are ready for the error trap code which begins with the Err_Label (with its colon). I have found this a good application for the Select Case structure. For the error trap I use Err, the error code number, as the expression. For the bare minimum trap I begin with only a Case Else statement and MsgBox, until I find out what special error handling I want to use. This simple error trap prevents the application’s giving a terse message (eg: “Object variable not Set”) and exiting. This can be a real help in working with a client who discovers “unexpected features” in your application. I have started including the procedure name in the message with the error number, Err, and the error text, Error(Err).

You can provide a more informative message or none at all.

```
Case 3078: MsgBox Err & ": " & Error(Err) & " Could not find Table: " &
s_table, MB_ICONSTOP ,"Procedure_Name FATAL ERROR"
End     'End Application
```

After the message it is time to specify what to do next. Usually it is “Resume Next” (the line of code that follows the one causing the error). Another choice is “Resume Exit_label”. While in the debugging stage, I have used “Resume 0”, that returns you to the offending line of code, which you can modify and try again. If you are trapping a fatal error, you can include an icon in the MsgBox and instead of “Resuming”, use the “End” statement to exit the application.

You can put any code such as additional logic with custom messages to provide appropriate detailed information

```
Case 3024: ' MsgBox "ERROR 3024: Could not find File: " & FileTitle$
   Call Create_new_DB(FileTitle$)
   Resume Open_DB_API
```

In this example I “commented out” the message and called another procedure, then resumed at a specific point in the original procedure.

```
MsgBox Err & ": " & Error(Err) & " occurred.",,"InsStrBfr ERROR"

    If (LocBefore <= 0) Then

        MsgBox "'" & BeforeString & "' not found in " & Chr(13) & "'" &
BaseString & "'"

    End If
    InsStrBfr = BaseString
    Resume Exit_InsStrBfr
```

There is a lot you can control with error traps!