Error Handling

When AMOS Professional encounters an error in your programs, or if you attempt the impossible, automatic
assistance is offered in the form of error messages displayed in the information line. If this happens while you are
programming, you can try to correct the mistake immediately. If the problem is found when you try to test or run a
program, AMOS Professional will take you to the offending line as soon as you edit.

Trapping errors

Routines can be set up in advance for handling errors inside an AMOS Professional program, so that when a mistake
is spotted, error trapping swings into action. This process is triggered by the following command:

ON ERROR

structure: trap an error within a Basic program

```
On Error Goto label
```

By this method, when an error occurs in your Basic program, a jump is made to whatever label has been defined.
This label acts as the starting point for your own error correction routine, and after the mistake has been corrected
you can return to your main program without the need to go via the editor window. Try this simple routine:

```
Do
Input "Type in two numbers";A,B
Print A;" divided by ";B;" is ";A/B
Loop
```

This will work perfectly until you try to enter a value of zero for B and it is discovered that division by zero is
impossible. Such unforeseen problems can be catered for by setting an error trap like this:

```
On Error Goto HELP
AGAIN:
Do
Input "Type in two numbers";A,B
Print A;" divided by ";B;" is ";A/B
Loop
Rem Error Handler
HELP:
Print
Print "Sorry, you have tried to divide"
Print "your number by zero."
Resume AGAIN : Rem Go back to input
```

If you are unfortunate enough to write an error inside your own error trapping routine, AMOS Professional will
grind to a halt! There are two ways to deliberately disable ON ERROR GOTO.

```
On Error : Rem disable error trap
```

Call ON ERROR without any parameters like that, or force it to go to zero, like this:

```
On Error Goto 0
```

To get back to your program after ON ERROR has been called, you must use RESUME. Never use GOTO for this
purpose.

RESUME

structure: resume the execution of current program after an error trapping routine

```
Resume
Resume Next
Resume labelname
Resume linenumber
```

Used on its own, RESUME will jump back to the statement which caused the error and try it again. To return to the
instruction immediately after the one that caused the error, use RESUME NEXT. Alternatively, to jump to a specific
point in your main program, simply follow RESUME with a reference to a chosen label or a normal line number.

ON ERROR PROC

structure: trap an error using a procedure

```
On Error Proc name
```

Errors can also be trapped using a procedure. ON ERROR PROC selects a named procedure which is automatically
called if there is an error in the main program. In fact, this is a structured version of the ON ERROR GOTO
command. In this case, your procedure must be terminated by an END PROC in the usual way, then return to the
main program with an additional call to RESUME, which can be placed just before the final END PROC statement.
Here is an example:

```
On Error Proc HELP
Do
Input "Type in two numbers";A,B
Print A;" divided by ";B;" is ";A/B
Loop
Rem Error Handler
Procedure HELP
Print
Print "Sorry, you have tried to divide"
Print "your number by zero."
Resume Next: Rem Go back to input
End Proc
```

When using a procedure to deal with errors, and you want to jump to a particular label, a special marker must be
placed inside that procedure. This is achieved with the RESUME LABEL structure.

RESUME LABEL

structure: jump to a label after an error has been isolated using a procedure

```
Resume Label label
```

This defines the label which is to be returned to after an error. It must be called outside of your error handler,
immediately after the original ON ERROR PROC or ON ERROR GOTO statement. For an example of RESUME
LABEL, please see the last routine in this Chapter.

ERRN

function: return the error code number of the last error

```
number=Errn
Print Errn
```

When you use ON ERROR to create error handling routines, you will want to know exactly what sort of error has
happened in the main program. Errors discovered while your program is running each have a specific error code
number, and the number of the last error to be isolated can be returned by using the ERRN function.

ERROR

instruction: deliberately generate an error and return to Editor

```
Error number
```

Supposing you have set up an error handling routine and you want to test your programming skills. The ERROR
command offers a simple method of simulating various mistakes without all the inconvenience of waiting for them

to happen. To test this system, select the error of your choice using the error code numbers listed in the next
Chapter. For example:

```
Error 88
```

That will quit your program and display a Disc full error message, simulating what would happen when your current
disc gets filled with data. You may also combine ERROR with the ERRN function, to print out the current error
condition, after a problem in your program:

```
Error Errn
```

Finally, ERROR can be used with RESUME LABEL inside an error handling routine, to jump straight back to a
label set up with the previous command. For example:

```
On Error Proc HELP
Resume Label WELCOME
Error 88
Print "This line is never printed"
WELCOME : Print "Hello! Hello! I'm back again!"
End
Procedure HELP
Print "There seems to be an error!"
Resume Label
Endproc
```

TRAP

instruction: trap an error

```
Trap instruction
```

The TRAP command offers a much sleeker error-trapping service than an ON ERROR GOTO structure, and it is
used to detect errors for a particular instruction. The TRAP instruction is used before any normal AMOS
Professional instruction with no colon between them, and it disables the error system for the specified instruction.
This means that if an error occurs, the program will not be halted, but the number of the error will be reported
instead. This number can then be returned by the ERRTRAP function, explained below. Here is an example of
trapping a disc access error:

```
Trap Load "File.Abk",10
If Errtrap : Print "Disc Error!" : End If
```

TRAP will only detect an error for the instruction that immediately follows, so in the next example the second usage
of the LOCATE instruction will cause an error!

```
Trap Locate -1,-1 : Locate -1,-1
```

ERRTRAP

function: return an error code number after a TRAP

```
number=Errtrap
```

This function is used to return the error status after a previous TRAP command. If no error has been detected, a zero
is returned, otherwise the appropriate error number is given. The related error message can then be returned using
the ERR$ function, explained next.

ERR$

function: return an error message string

```
text$=Err$(error number)
```

This simple function returns an error message string. If the error number is out of range, then an empty string will
be given. ERR$ will return error messages as long as they are loaded in memory, but messages will not be returned
from a compiled program, or if the Editor has been KILLed.