ON ERROR Capability - an enhancement to
BASIC/3000 and BASICOMP/3000.

As part of Hewlett Packard's ongoing efforts to improve current products, BASIC and BASICOMP/3000 will be enhanced to provide user error-handling capabilities.

This article describes what ON-ERROR will look like as well as what ON-ERROR will do for the BASIC/3000 user.

The ON ERROR enhancement to the BASIC subsystems will

1. Allow suppression of BASIC RUN-TIME ERROR messages.

2. Cause an automatic branch to a USER-SUPPLIED Run-Time Error routine when desired.

3. Capture and display information as to the nature of the Run-Time Error. (Error Number and Error Message)

4. Capture information as to the Line Number of the Basic program which caused the Error to occur.

REASONS FOR DEVELOPING THIS ENHANCEMENT

Any application written in BASIC/3000 can be intended to run in an environment which is not a sophisticated programming environment. Unsophisticated users need to be buffered from system messages which they may not understand.

Many of our BASIC/3000 customers are OEM's.
They are sophisticated BASIC software shops who understand the 3000 SYSTEM, and appreciate the Basic internal error messages and/or system information when things go wrong during the development phase.
However, the point at which they release their software to their customers is the point at which they no longer want arbitrary error messages coming out.
They need an ON-ERROR capability which allows them to buffer their customers in a friendly manner. As an extension of that capability, the OEM needs "hooks" to allow access to the Basic internal and/or system error information

so that what went wrong in the customer's environment can be understood.

Thus, the user sees ON-ERROR capability as a way to aid him in his development of BASIC programs.

He also sees ON-ERROR as a way to buffer the unsophisticated users of his software from unsolicited system information.

FUNCTIONAL DESCRIPTION OF ON-ERROR

The user will have three ON-ERROR options:

1. ON ERROR [THEN] <LABEL>
   [GOTO]

2. ON ERROR GOSUB <LABEL>

3. ON ERROR CALL <SUBPROGRAM>

OFF ERROR will restore normal system error handling to a BASIC program.

The Built-in Function ERRL will return the line number of the statement where the error was detected.

The Built-in Function ERRN will return the Basic or Basicomp error number of the error that occurred.

The Built-in Function ERR$ will translate an error number to an ENGLISH, printable explanation string.

IMPLEMENTATION DETAILS OF ON-ERROR

Compiled BASIC programs:

a) "For-Next" internal structure will be removed from the stack only for the "ON-ERROR GOTO" statement. Thus, if user is within a for-next loop, and an ON-ERROR GOSUB or an ON-ERROR CALL statement occurs within the scope of that for-next loop, and a run-time error occurs, the for-next information will only be removed from the stack opon return from the gosub or the call and after the for-next loop is satisfied.

b) When an ON-ERROR GOSUB or an ON-ERROR CALL occurs, upon returning, the user will return to the beginning of the next BASIC statement after the BASIC statement which caused the run-time error.

c) Code generation for flavors of the ON-ERROR are as followa:

1) ON ERROR GOTO <LABEL>

```
BR N (Branch around the code for the "goto")
R:  code for GOTO
    .
    .
    .
N:  code for statement following ON-ERROR statement
```

****
Code will also be generated to place the P-relative location for "R" into the "ON-ERROR GOTO" runtime global.
****

2) ON ERROR GOSUB <LABEL>

```
BR N (Branch around the code for "gosub")

R:  Code for GOSUB
    .
    .
    .
N:  code for statement following ON-ERROR statement
```

****
Code will be generated to place the P-relative location for "R" into the "ON-ERROR GOSUB" runtime global.

Code within each "RETURN" statement will check whether or not this return is within scope of an ON ERROR GOSUB. If so, RETURN will be to statement following the statement which caused the runtime error.
****

3) ON ERROR CALL <SUBPROGRAM>

```
BR N (Branch around the code for "call")

R:  Code for CALL
    .
    .
    .
```

Code to branch to P-relative location
following the place where the run-time
error was encountered will be generated.

N: code for statement following ON-ERROR statement

****
Code will be generated to place the P-relative location
for "R" into the "ON-ERROR CALL" runtime global.
****

d) No ON-ERROR statement will be allowed within
the scope of an ON-ERROR GOSUB subroutine.

Interpreted BASIC programs:

a) When an ON-ERROR GOSUB or an ON-ERROR CALL occurs,
upon returning, the user will return to the beginning
of the next BASIC statement after the BASIC statement
which caused the run-time error.

b) No ON-ERROR statement will be allowed within
the scope of an ON-ERROR GOSUB subroutine.

IMPLEMENTATION DETAILS OF OFF-ERROR

When an ON-ERROR statement is implemented, a global
is updated to describe the type of on-error statement
encountered. At the time a run-time error occurs, all
routines which relate to on-error will consult that
global.
If that global has a value of zero, it is assumed that
there is no ON-ERROR statement in the program.
There may be instances where a user wants on-error
capability only for certain parts of his/her program.
Thus, the OFF-ERROR statement will reset the on-error
global until another ON-ERROR statement is encountered.

NEW BUILT-IN FUNCTIONS

a) ERRN in Interpreter and Compiler

The Built-in Function "ERRN" will return the error number of the last runtime error which occurred in the program.

It should be noted that if, prior to the occurrance of this run-time error, there was no ON-ERROR statement in the program, it is highly likely that the run-time error will cause a program abort, and "ERRN" will be irrelevent.

b) ERR$ in Interpreter and Compiler

The Built-in Function ERR$ will translate an error number to an ENGLISH, printable string.

It should be noted that if, prior to the occurrance of this run-time error, there was no ON-ERROR statement in the program, it is highly likely that the run-time error will cause a program abort, and "ERR$" will be irrelevent.

c) ERRL in Interpreter and Compiler

ERRL will return the line number in which the offending run-time error occurred.

It should be noted that if, prior to the occurance of this run-time error, there was no ON-ERROR statement in the program, it is highly likely that the run-time error will cause a program abort, and "ERRL" will be irrelevant.

ON-ERROR VS ON-END

A BASIC program can contain both ON-END and ON-ERROR statements simultaneously.

The following truth table defines all possibilities which can occur within a program when a run-time error "END OF FILE" occurs:

ON-END FOR FILE N	ON-ERROR	WHAT HAPPENS
0	0	PROGRAM DIES
0	1	ON-ERROR
1	0	ON-END
1	1	ON-END

PERFORMANCE OBJECTIVES

For the most part, ON-ERROR capability will not entail a performance cost.

ERRN will not impact performance.

In the BASIC INTERPRETER, the function ERRL will have no detrimental performance effects. A Basic Compiled program, however, will suffer a performance degradation with ERRL in that the code will be blown up to include a "label table" to be consulted at Runtime.

To ensure that only those who need ERRL will have to pay for it, ERRL will be a Compiler Option. Default for ERRL option will be OFF.

To turn on ERRL, use $CONTROL ERRL.

SUMMARY

The ON-ERROR enhancement to BASIC/BASICOMP 3000 is a major enhancement consisting of three new statements and three new built-in functions. The user will be able to turn ON-ERROR on and off during the course of any one run of a BASIC program. He will also be able to obtain the Line Number of where a run-time error occurred, Error Number of the run-time error, and the actual Error Message which would have come out had the ON-ERROR statement not been in effect.