BULLETPROOF ERROR TRAPPING

Learn how Applesoft's powerful error trapping system works and how to use it effectively in your programs.

Applesoft's error trapping system is one of its most powerful features. It can keep a program running under the most trying circumstances. In today's world of bulletproof and user-friendly programs, every programmer should know how it works.

ERROR TRAPPING COMPONENTS

ONERR *line number* turns on the error trap system. Before this statement is encountered in the program flow, most errors bring the program to an abrupt halt with a sounding of the bell and a message like I/O ERROR IN 2530.

With the error trap in place, this type of error causes the program to branch to *line number*, where the error can be handled. In the case of an I/O error, the error handling part of the program would probably tell you to check the drive door and try again.

POKE 216,0 turns off error trapping. You may want to use error trapping only when disk operations are taking place, letting the program stop on other kinds of errors. It is also common to turn off error trapping when an error has occurred so that a subsequent error, such as pressing Control-C, doesn't yield misleading information.

It can keep a program running under the most trying circumstances.

PEEK(222) tells you which error has occurred. See **Table 1** for a list of the error numbers and their meanings.

PEEK(218)+256*PEEK(219) tells you the program line where the error was encountered. When a disk error occurs, it usually doesn't matter where in the program the error occurred. However, if it is a syntax error, which could be caused by the user mistyping the program from a magazine listing, the line number helps in debugging.

RESUME restarts the program at the point where the error occurred. While this command may seem really handy, there are actually few instances where you would want to do this (as demonstrated later), so RESUME is seldom used.

The programs are compatible with both DOS 3.3 and ProDOS.

TABLE 1: DOS 3.3, ProDOS and Applesoft Error Messages

Number	Error Message	Source
0	NEXT WITHOUT FOR	Applesoft
1	LANGUAGE NOT AVAILABLE	DOS 3.3
2	RANGE ERROR	ProDOS
3	NO DEVICE CONNECTED	ProDOS
4	WRITE PROTECTED	DOS/ProDOS
5	END OF DATA	DOS/ProDOS
6	PATH NOT FOUND	ProDOS
	FILE NOT FOUND	DOS 3.3
7	PATH NOT FOUND	ProDOS
	VOLUME MISMATCH	DOS 3.3
8	I/O ERROR	DOS/ProDOS
9	DISK FULL	DOS/ProDOS
10	FILE LOCKED	DOS/ProDOS
11	INVALID PARAMETER	ProDOS
	SYNTAX ERROR	DOS 3.3
12	NO BUFFERS AVAILABLE	DOS/ProDOS
13	FILE TYPE MISMATCH	DOS/ProDOS
14	PROGRAM TOO LARGE	DOS/ProDOS
15	NOT DIRECT COMMAND	DOS/ProDOS
16	SYNTAX ERROR	Applesoft
17	DIRECTORY FULL	ProDOS
18	FILE NOT OPEN	ProDOS
19	DUPLICATE FILE NAME	ProDOS
20	FILE BUSY	ProDOS
21	FILE(S) STILL OPEN	ProDOS
22	RETURN WITHOUT GOSUB	Applesoft
42	OUT OF DATA	Applesoft
53	ILLEGAL QUANTITY	Applesoft
69	OVERFLOW	Applesoft
77	OUT OF MEMORY	Applesoft
90	UNDEF'D STATEMENT	Applesoft
107	BAD SUBSCRIPT	Applesoft
120	REDIM'D ARRAY	Applesoft
133	DIVISION BY ZERO	Applesoft
163	TYPE MISMATCH	Applesoft
176	STRING TOO LONG	Applesoft
191	FORMULA TOO COMPLEX	Applesoft
224	UNDEF'D FUNCTION	Applesoft
254	Bad response to INPUT	Applesoft
255	Control-C interrupt attempt	Applesoft

CALL −3288 fixes up the stack so that a program can continue without a RESUME statement. The stack is a special 256-byte area of memory used (among many other uses) by BASIC to keep track of FOR-NEXT loops and GOSUB-RETURN combinations. When a branch occurs as the result of an ONERR trap, information about where the error occurred is left on the stack. If this information is allowed to remain on the stack, it will eventually cause problems.

Early Applesoft manuals provided a short machine language program that could be POKEd into page 3 and then CALLed at the proper times. Then someone discovered that this code already existed within the Monitor ROM code.

Many programmers rely too much on an ONERR trap, expecting it to do too much.

DEVELOPING AN ERROR TRAP

Listing 1 is a simple program that lets you type in the names of Hi-Res picture files and view them on page 1. Make sure you have a disk with a Hi-Res picture file on it on hand. If you don't have a Hi-Res picture file, you can create a simple one with the following procedure:

```
HGR: HCOLOR=3: HPLOT 10,10 TO 100,100: TEXT
BSAVE PICTURE,A$2000,L$2000
```

Then run the program to see the error trap in action. First, try typing a name that doesn't correspond to a file on the disk. The program looks for the file on the disk and then reports its failure by printing the appropriate message. You can demonstrate other errors by leaving the drive door open, typing a name that begins with a number, and typing the name of a program (type A) file. In each case, an appropriate message is displayed, and when you press Return, you have another chance to correct your mistake. **Lines 150 and 180** report errors that would only occur if you were trying to save files on the disk, but they've been included to provide a more general routine. If your program manipulates text files, you'll need a message (and perhaps some additional code) for error number 5. If your program involves calculations with user-supplied data, you may need program lines that can handle errors 53, 69, and 133. These specific applications are beyond the scope of this article.

Line 80 turns on the error trap by directing flow (on an error) to the code beginning at **line 130**. In **line 130**, error trapping is turned off with a POKE 216,0. The error number and the line number where it occurred are then read into the variables E and EL. Finally, the stack is prepared for a non-RESUME exit with a CALL −3288.

Lines 150-200 check E for certain error codes and print the appropriate message. If none of these codes is found, **line 210** prints a general message, which includes the error number and line number.

Lines 220-240 offer a choice of continuing or quitting. To continue, the program must branch back to **line 80**, where the error trap is reinstated.

If your program uses the printer, it's also a good idea to include a PRINT CHR$(4)"PR#0" statement near the beginning of the error handling code. This prevents the error messages from appearing on the printer.

COMMON MISTAKES

As simple as this process may seem, there's plenty of room for error. Here are several of the mistakes that can cause trouble. Some of them are hard to detect.

1. *Placing statements after the ONERR statement on the same line.* An ONERR statement must be the last statement on a line. You can demonstrate this for yourself by placing a GOTO 240 statement at the end of **line 80**. If the statement were recognized, the program would end immediately. It is ignored, however, and the program continues about its business.
2. *Failure to use a CALL −3288.* This is particularly sneaky. In many cases, you won't encounter a problem. However, if your error occurs in the middle of a subroutine, you'll surely get a RETURN WITHOUT GOSUB ERROR for no apparent reason. Try removing the CALL −3288 from **line 260** of **Listing 2** to see what happens when it's missing. Even in a program such as **Listing 1** (try removing the CALL −3288 from **line 130**), repeated errors and restarts will eventually result in an OUT OF MEMORY ERROR. (The memory that's run out is the 256-byte stack, not the 32K of Applesoft program memory.) Even though this kind of deliberate program abuse is very unlikely, the program is not bulletproof.
3. *Misusing RESUME.* There are few circumstances where you need or want to use it. An example is described below.

   To demonstrate a misuse of RESUME, remove the CALL −3288 and POKE 216,0 from **line 130** and change **line 230** so that it reads:

   ```
   230 GET Z$: PRINT: IF Z$<>CHR$(27) THEN RESUME
   ```

   When you run the program, type a name that doesn't correspond to a file on the disk. The FILE NOT FOUND message appears, but when you press Return to try again, the program immediately looks for the nonexistent file again. You're caught in a loop where the only exit is to stop the program. You have to be able to type in the correct file name, and RESUME won't let you do it.

   By the way, one of the few instances in which you might want to use RESUME is a printer code input routine, where you want Control-C to be a valid input character. In this case, you want to test for error 255 first and execute a RESUME before the program has a chance to turn the error trap off or to execute CALL −3288.
4. *Tying actions to line numbers.* This error occurs in more complex programs, where different types of errors occur in different parts of the program. Many programmers use the line number information reported by PEEK(218)+256*PEEK(219) to identify the part of the program where the error occurred. This works fine until the program gets modified and renumbered. The statement IF EL=2190 THEN GOTO 3000 may no longer identify the proper part of the program. This is because the 2190 part of the statement doesn't get renumbered. A more flexible system using a flag variable is demonstrated in **Listing 2**.
5. *Forgetting to turn the error trap back on.* To demonstrate this, just change the GOTO 80 in **line 230** of **Listing 1** to GOTO 90. The first time you encounter an error, the error trap is invoked; the second time, the program crashes.

COMPLEX ERROR TRAPPING METHODS

Listing 2 demonstrates a simple method of tracking where a program error occurs, without relying on line number comparisons. The program is similar to **Listing 1**, but it also offers a disk catalog and file save option.

The use of the variable EF has been added to the error trapping scheme. At the beginning of each ONERR line, EF is set to the value 1, 2 or 3. **Line 360** uses this value to direct flow back to the proper part of the program after an error has occurred.

PREVENTION

Many programmers rely too much on an ONERR trap, expecting it to do too much. Other techniques for handling errors don't involve the ONERR construction. Instead, they're specifically directed at keeping the program out of the error trap in the first place.

Range Checking

Don't let the user of your program enter numbers that will cause an error. If an index into an array is entered, make sure the user has to enter a legal subscript for the array. Similarly, don't let the user enter numbers that will result in dividing by zero or in taking the square root or logarithm of a negative number.

Type Conversion

Never use a numeric variable as the input parameter of an INPUT or GET statement. All the user has to do is press Return without entering any data to blow it up. Instead, take all input into a string variable and then convert it to a numeric variable.

Sometimes it takes a few program lines to avoid an error. For instance, you could check the length of file names under ProDOS and make the user retype any names over 15 characters that are entered. You could go one step further and check that the name contains only letters, numbers and periods, and that it begins with a letter. The following program segment does it all. You may want to use a variation of it in your own programs.

```
10 REM PRODOS FILE NAME CHECKER
20 INPUT "FILE:";F$:NF = 1: IF LEN (F$) > 0 THEN FOR
   NC = 1 TO LEN (F$):NA = ASC ( MID$ (F$,NC,1)):
   NF = ((NA > 64 AND NA < 91) OR (NA = 46) OR (NA >
   47 AND NA < 58 AND NC > 1)) AND NF = 1: NEXT : IF
   LEN (F$) > 15 OR NF = 0 THEN PRINT "INVALID
   NAME": GOTO 20
30 PRINT "VALID NAME"
```

LISTING 1: ONERR.EX1

```
10   REM *********************
20   REM *      ONERR.EX1     *
30   REM *   BY LOREN WRIGHT  *
40   REM * COPYRIGHT (C) 1987 *
50   REM * BY MICROSPARC, INC *
60   REM * CONCORD, MA  01742 *
70   REM *********************
80   ONERR  GOTO 130
90   HOME : TEXT : VTAB 12: INPUT "FILE NAME:
      ";NA$
100   HGR
110   PRINT  CHR$ (4)"BLOAD"NA$",A$2000"
120   VTAB 22: PRINT "PRESS RETURN TO CONTINUE
      ";: GET Z$: PRINT : GOTO 90
130   POKE 216,0:E =  PEEK (222):EL =  PEEK (2
      18) + 256 *  PEEK (219): CALL  - 3288
140   HOME : TEXT : VTAB 12
150   IF E = 4 THEN  PRINT "DISK IS WRITE-PROT
      ECTED": GOTO 220
160   IF E = 6 THEN  PRINT "FILE NOT FOUND ON
      THIS DISK": GOTO 220
170   IF E = 8 THEN  PRINT "I/O ERROR--CHECK D
      RIVE DOOR": GOTO 220
180   IF E = 9 THEN  PRINT "DISK FULL": GOTO 2
      20
190   IF E = 11 OR (E = 16 AND  PEEK (48896) =
      76) THEN  PRINT "ILLEGAL FILE NAME": GOTO
      220
200   IF E = 13 THEN  PRINT "FILE TYPE MISMATC
      H": GOTO 220
210   PRINT "ERROR "E" IN LINE "EL
220   VTAB 22: HTAB 4: PRINT "RETURN TO CONTIN
      UE, ESCAPE TO QUIT";
230   GET Z$: PRINT : IF Z$ <  >  CHR$ (27) GOTO
      80
240   END
```

END OF LISTING 1

```
              KEY PERFECT 5.0
                  RUN ON
                ONERR.EX1
=======================================
  CODE-5.0    LINE# - LINE#    CODE-4.0
  --------    -------------    --------
  30B3CD7D       10 -   100      593E
  6266A807      110 -   200      ACD3
  A9FB20DC      210 -   240      2DA0
  44481ECA = PROGRAM TOTAL =     028F
```

LISTING 2: ONERR.EX2

```
10   REM *********************
20   REM *      ONERR.EX2     *
30   REM *   BY LOREN WRIGHT  *
40   REM * COPYRIGHT (C) 1987 *
50   REM * BY MICROSPARC, INC *
60   REM * CONCORD, MA  01742 *
70   REM *********************
80   REM  MAIN MENU
90   HOME : TEXT : VTAB 9: PRINT "1) LOAD FILE
      ": PRINT : PRINT "2) SAVE FILE": PRINT :
      PRINT "3) CATALOG": PRINT : PRINT "4) Q
     UIT"
100   VTAB 18: PRINT "ENTER NUMBER OF CHOICE:
      ";: GET Z$: PRINT : IF Z$ < "1" OR Z$ >
      "4" THEN  PRINT  CHR$ (7): GOTO 100
110   ON  VAL (Z$) GOSUB 120,170,210,240: GOTO
      90
120 EF = 1: ONERR  GOTO 260
130   HOME : TEXT : VTAB 4: HTAB 15: PRINT "LO
      AD FILE": VTAB 22: HTAB 12: PRINT "RETUR
      N FOR MENU": VTAB 12: HTAB 1: INPUT "FIL
      E NAME: ";NA$: IF NA$ = "" THEN  RETURN
140   HOME : HGR
150   PRINT  CHR$ (4)"BLOAD"NA$",A$2000"
160   VTAB 22: PRINT "PRESS RETURN TO CONTINUE
      ";: GET Z$: PRINT : RETURN
170 EF = 2: ONERR  GOTO 260
180   HOME : TEXT : VTAB 4: HTAB 15: PRINT "SA
      VE FILE": VTAB 22: HTAB 12: PRINT "RETUR
      N FOR MENU": VTAB 12: HTAB 1: INPUT "FIL
      E NAME: ";NA$: IF NA$ = "" THEN  RETURN
190   PRINT  CHR$ (4)"BSAVE"NA$",A$2000,L$2000
      "
200   HOME : VTAB 22: PRINT "PRESS RETURN TO C
      ONTINUE";: GET Z$: PRINT : RETURN
210 EF = 3: ONERR  GOTO 260
220   PRINT  CHR$ (4) LEFT$ ("CATALOG",7 - 4 *
      ( PEEK (48896) = 76))
230   PRINT "PRESS RETURN TO CONTINUE";: GET Z
      $: PRINT : RETURN
240   HOME : VTAB 12: INPUT "ARE YOU SURE YOU
      WANT TO QUIT? ";YN$: IF YN$ <  > "Y" AND
      YN$ <  >  CHR$ (121) THEN  RETURN
250   END
260   POKE 216,0:E =  PEEK (222):EL =  PEEK (2
      18) + 256 *  PEEK (219): CALL  - 3288
270   HOME : TEXT : VTAB 12
280   IF E = 4 THEN  PRINT "DISK IS WRITE-PROT
      ECTED": GOTO 350
290   IF E = 6 THEN  PRINT "FILE NOT FOUND ON
      THIS DISK": GOTO 350
300   IF E = 8 THEN  PRINT "I/O ERROR--CHECK D
      RIVE DOOR": GOTO 350
310   IF E = 9 THEN  PRINT "DISK FULL": GOTO 3
      50
320   IF E = 11 OR (E = 16 AND  PEEK (48896) =
      76) THEN  PRINT "ILLEGAL FILE NAME": GOTO
      350
330   IF E = 13 THEN  PRINT "FILE TYPE MISMATC
      H": GOTO 350
340   PRINT "ERROR "E" IN LINE "EL
350   VTAB 22: HTAB 4: PRINT "RETURN TO CONTIN
      UE, ESCAPE TO QUIT";
360   GET Z$: PRINT : IF Z$ <  >  CHR$ (27) THEN
       ON EF GOTO 120,170,210
370   END
```

END OF LISTING 2

```
              KEY PERFECT 5.0
                  RUN ON
                ONERR.EX2
=======================================
  CODE-5.0    LINE# - LINE#    CODE-4.0
  --------    -------------    --------
  B1348AB1       10 -   100      952B
  FB7E9548      110 -   200      A85C
  11B6FC7A      210 -   300      9666
  E5EF6701      310 -   370      64A6
  E1B95E3B = PROGRAM TOTAL =     04D9
```