

Extended BASIC change/error
list and their Dispositions
for 99/4 & 99/4A users

Updated 9/4/81

The CHANGE or ERROR indicated below is either something that a majority of users do not like or an error in the current version of Extended BASIC interpreter.

The fixed date showing below is the date concerning software release, not for the production. It takes at least 12 weeks to get the new semiconductor chip for the command module. Therefore, the new versions of Extended BASIC probably are still not available at this time.

***** 01 *****

CHANGE:

ACCEPT AT(1,1) SIZE(6):X\$

When in the last position of a fixed length input, an error beep is sounded. This is very annoying - especially when entering a legal character.

STATEMENT:

This has been changed 7/02/81. In the new version, a tone is sounded only when the user attempts to enter beyond the field boundary. In the current version, the error tone is intended to inform the user that they are at the last input position. If this is annoying then the only recourse is to turn down the volume.

***** 02 *****

ERROR:

Adding a keyword to a line which has already filled the entire screen makes changes on the screen and crashes the system.

Example:

10 GOTO GOTO GOTO GOTO GOTO GOTO GOTO

A keyword has to be 4 or more characters long.

STATEMENT:

Only intentional meaningless input should cause this problem, there is no correction planned for it.

***** 03 *****

ERROR:

Illegal line number 0 can be created by editing a line.

Example: 1. Create a line 10 PRINT

2. Press 10 and-down arrow to bring up the line.

3. Press shift-R and get 10 with no statement.

4. Delete line number 10 and type in PRINT.

5. Then LIST will list 0 PRINT

10 PRINT.

STATEMENT:

This problem has been fixed 7/02/81.

In the current version, if line number zero is accidentally created, RESEQUENCE and delete the first line (old line zero).

***** 04 *****

ERROR:

When an error occurs in an argument list in a CALL statement, and by ON ERROR the program execution continues and the same subprogram is called again RECURSIVE SUBPROGRAM error is issued.

Example: 10 ON ERROR 40
20 CALL X(VAL("S"))
30 STOP
40 RETURN 20
50 SUB X(N)::SUBEND

STATEMENT:

This problem has been fixed 7/02/81. In the current version, there is no known way of programming around this problem. Try to limit the number of expression that are calculated at RUN time in a CALL statement, i.e.,
>20 TEMP=VAL("S")
>25 CALL X(TEMP)

***** 05 *****

CHANGE:

PRINT statement in EXTENDED BASIC clears a line before scrolling the line. Incompatible with Home Computer BASIC.

STATEMENT:

This incompatibility with 99/4 BASIC in the PRINT will not be fixed. There is a reason for both to be as they are.

***** 06 *****

CHANGE:

A large program file that was created and saved by the console BASIC may not be loaded in Extended BASIC and may crash system. This may happen even if Expansion memory is attached.

This is due to the fact that:

- (1) A larger amount of VDP RAM memory is available for the console BASIC than for the extended BASIC.
- (2) All console BASIC programs are stored in a program format whereas the extended BASIC will save its file in a sequential file format, variable length max. 254 if the program size exceeds VDP RAM space and Expansion Memory is attached.
- (3) Program format files must be loaded into VDP RAM before Extended BASIC can transfer them to Expansion Memory.

STATEMENT:

If a program file is too large for the Extended BASIC, save the original program and create a second copy and delete several lines from the second. Keep eliminating lines until the file is small enough to fit in the VDP RAM space available for the extended BASIC.

Edit the file on extended BASIC by adding the lines previously deleted. Then when the program is saved, it will be stored as a sequential file that Extended BASIC can load directly into Expansion Memory.

***** 07 *****

ERROR:

CLEAR breaks a program during an execution of INPUT, ACCEPT or LINPUT statement regardless of ON BREAK NEXT flag.

STATEMENT:

This problem has been fixed 7/02/81. There is no known way of programming around this problem. Care must be taken when pressing CLEAR.

***** 08 *****

ERROR:

MEMORY FULL error occurring during MERGE execution will leave the file open to disk DSR (Device Service Routine).

Example : > CALL FILES(1)
> NEW
> OLD DSK1.TEST
> MERGE DSK1.TEST2
* MEMORY FULL
> SAVE DSK1.SFILE
* I/O ERROR 64

When memory full error occurs you can not LOAD, SAVE or open another file.

STATEMENT:

This has been fixed 07/02/81, however, in the current version, to avoid this problem, always make sufficient number of files available to disk DSR before merging a file. In the above example CALL FILES(2) before merging a file will solve the problem.

If you do not care if you lose the file in the memory, simply do NEW and proceed.

***** 09 *****

ERROR:

```
100 DIM S(1720)           (This example only applies to the 99/4
110 A$=""                 console with no peripheral attached to it)
120 A$="HELLO THERE COMPUTER"
130 PRINT A$
140 A$=A$
150 GOTO 130
```

Assigning a string to itself (A\$=A\$) with memory full, the string may get garbled or destroyed by garbage collection during the string assignment.

STATEMENT:

This has been fixed 07/02/81. In the current version, however, alter the size of the program (or array if there is one) a little, the problem should go away. If possible, avoid assigning a string variable to itself.

***** 10 *****

ERROR:

A garbage collection which occurs during a PRINT USING statement execution may occasionally cause a wrong output. The particular example that caused this problem was:

```
100 DIM S(1695)           ( No disk, no memory expansion.
110 IMAGE #####          With disk modify line 100 to
#####                 DIM S(1434). )
120 A=100000::B=222222
130 A$="OIOIOIOIO"
140 B$="HELLO"
150 PRINT USING 110:A,A$,B,B$
160 GOTO 120
```

STATEMENT:

This has been fixed 2/24/81. In the current version, when this problem is discovered alter the size of the program slightly and run the program again. This problem should be eliminated. If a MEMORY FULL error is issued after the program size has been changed, make your program smaller by cutting an array size, eliminating unnecessary variables, making multiple statement lines, etc.

The same problem may be noted when you try to allocate a speech string if a memory is full, even though this type of problem rarely occurs.

***** 11 *****

ERROR:

A long constant in a variable field in INPUT, ACCEPT, LINPUT, READ, and NEXT may crash the system. This most often happens when a programmer accidentally uses a wrong delimiter after an INPUT prompt. Example: 10 INPUT "1234567890123456789012345678901234567890",A

STATEMENT:

This has been fixed 7/02/81. In the current version, when using a long constant in INPUT or LINPUT statement make sure your program in the line is grammatically correct. Especially be sure that a colon is placed after an INPUT or LINPUT prompt.

***** 14 *****

ERROR:

If a PROTECTED program is in memory, a NUMBER command is issued, and the first line number generated is for a line already in the program, then Extended Basic goes into a loop displaying *PROTECTION VIOLATION and sounding the error tone. The only way out of the loop is the QUIT key.

STATEMENT:

NUM or NUMBER is an illegal command when a protected program is in memory, since it would list and/or alter parts of the program. If you use the command by accident, you must press QUIT. You will not have lost the program, since in order to have a protected program in memory, you must have loaded it from a storage device with the OLD command. Just reenter Extended Basic and reload the program. This has been fixed 07/02/81.

***** 15 *****

ERROR:

```
100 CALL A
110 SUB A
120 INPUT X
130 SUBEND
RUN
PPPP
```

```
* WARNING
  INPUT ERROR IN 120
  IN X
  CALLED FROM 28264
?
```

When INPUT statement (only under screen input case) is used inside a subprogram, invalid information may be shown in warning messages when illegal input is found.

STATEMENT:

Inside the warning message, the first line number where the INPUT statement appears is correct. Adding any input prompt (even " ") to that INPUT statement can avoid this problem.

***** 16 *****

ERROR:

```
100 OPEN #1: "DSK1. INPUT", FIXED 32, RELATIVE 5
110 CALL A
120 SUB A
130 PRINT #1: "HELLO"&CHR$(0)&"THERE"
140 INPUT #1, REC 0: A$
150 SUBEND
RUN
```

```
IN 140
IN A
CALLED FROM 110
```

When INPUT statement (only under file input case) is used inside a subprogram, "INPUT ERROR" may be missing from the error message.

STATEMENT:

Inside the warning message, the first line number can locate the error in a line which contains the INPUT statement.

***** 17 *****

ERROR:

Example A:

```
100 I=5
110 ACCEPT AT(3,4): A$(I+1)
120 END
```

Example B:

```
100 ACCEPT AT(3,4) VALIDATE(DIGIT)
      SIZE(3): A(V-K)
```

In a ACCEPT statement without VALIDATE clause, if an expression is used as the subscript of an array in the accept-item variable, then this statement may not accept any value except the enter key.

SIZE clause does not work even when VALIDATE clause is added. See Example B.

STATEMENT:

Use a VALIDATE clause inside the ACCEPT statement when an expression is used as the subscript of an array in the accept-item variable, the first problem should go away. No solution, at present, to the second problem.

***** 18 *****

ERROR:

If you get an error in an immediate command while a program is stopped at a breakpoint, a CONTINUE may not work properly.

Example:

```
>100 FOR I=1 TO 100 :: PRINT I :: NEXT I
>RUN
(break program with CLEAR)
:ON
* ONLY LEGAL IN A PROGRAM
:CON
(run to completion, but program is lost.)
```

STATEMENT:

If you get an error in an immediate command while your program is stopped at a breakpoint, be sure to issue an immediate command that does work before you continue program execution. For example,

```
>100 FOR I=1 TO 100 :: PRINT I :: NEXT I
:RUN
(break with CLEAR)
:ON
* ONLY LEGAL IN A PROGRAM
:PRINT
:CON
(program runs normally)
```

***** 19 *****

CHANGE: Faster Extended BASIC programs

If you have the Memory Expansion unit attached to your computer, you can increase the execution speed of your TI Extended BASIC programs. Simply include the statements

```
CALL INIT :: CALL LOAD [-31878,0]
```

near the beginning of your program. These statements disable all sprite motion. To move sprites, change the 0 in the CALL LOAD statement to a number from 1 TO 28, for the number of moving sprites with which you want to work. For example, the statement

```
CALL LOAD [-31878, 3]
```

allows Sprites 1, 2, and 3 to move. The CALL LOAD statement should be executed after any CALL DELSPRITE(ALL) statement and at the beginning of execution.