SECTION VII
Debugging

BASIC/3000 provides commands that allow a program to be debugged while it is running. The path of execution through a program and the change in value of variables can be traced. The dynamic nesting structure of a program can be displayed; variable values can be displayed and modified; tracing can be changed; and the execution sequence can be altered.

Note that once a program has been saved for RUNONLY (see Section II), it cannot be debugged.

TRACE/UNTRACE Commands

The TRACE command is used to turn on the tracing of selected variables, both simple variables and arrays, function references (with or without tracing their local variables), programs called with INVOKE or CHAIN, and statements of the current program. UNTRACE turns off tracing.

Form

The commands have these forms:

TRACE [trace element list]

UNTRACE [trace element list]

The *trace elements* are optional and include *variables* (including those local to functions), *functions*, *labels*, a range of labels *(label-label)*, and the keyword PROG.

Variables include simple variables, string variables, string arrays, and numeric arrays. Numeric arrays are distinguished from simple variables by a (*) or (*,*) following the array name. Examples of variables:

A, B(2)	numeric variable
B$, C$(*)	string variable
C$(*,*)	string array
A(*)	one-dimensional numeric array
A(*,*)	two-dimensional numeric array

Functions can be specified by the function name only or the name followed by a list of local variables in parentheses. For example:

FNA	numeric function name
FNB(A$,B(*))	local string variable and local numeric array; FNB is not traced, only the local variables.

Labels are statement labels consisting of integers in the range 1 through 9999.

Label-label stands for all statements between, and including the statements identified by the two labels. For example:

80	the statement at label 80
100-150	all statements between 100 and 150 inclusive

PROG is a keyword to specify that trace information be printed when a CHAIN or INVOKE is performed.

Explanation

The BASIC/3000 Interpreter keeps track of all items specified in TRACE commands. A range of labels traces all statements within the range. Any change to a variable's value, any reference to a function, or any execution of a statement causes tracing information to be printed. Any change in the value of an array element, except in MAT operations, causes that element to be printed. To trace a function, only the function name is specified. If a list of local variables is included with the function name, changes in their values are traced but not execution of the function. If PROG is specified in a TRACE command, information is printed whenever a program is accessed through an INVOKE or CHAIN statement and also when returning to an invoking program (see Section X, Segmentation).

TRACE with no parameters lists the items currently being traced.

UNTRACE turns off tracing of the items specified. When tracing is turned off for a function it is not turned off for any local variables. When tracing is turned off for variables individually, it does not turn off tracing for the function. For instance, UNTRACE FNA turns off tracing of the function but not of its local variables if they were specified in a TRACE command; UNTRACE FNA (A$) turns off tracing of the local variable A$ but not of the function FNA.

UNTRACE with no parameters stops all tracing specified by previous TRACE commands.

The specified tracing occurs only when the program is executed. When trace output occurs, the following information is printed:

@ *label variable = value*
or
@ *programname label variable = value*

These outputs are printed as a result of a trace of a function or variable.

The programname is printed if the current program has been named.

*TRACE *label*
or
*TRACE *programname label*

These outputs are printed as a result of a trace of a label or labels.

Examples

The program below includes a one-dimensional array, a simple variable, and a function call. This program will be used for succeeding TRACE and UNTRACE examples.

```
  10 DIM X[5]
  20 LET Q=5
  30 READ (FOR N=1 TO 5,X[N])
  40 DATA 2.5,75.6,36.2,15.7,100
  50 PRINT FND(X[*],Q)
 210 DEF FND(A[*],INTEGER N)
 220   REAL I,J
 230   J=1
 240   FOR I=2 TO N
 250     IF A[I]>A[J] THEN J=I
 260   NEXT I
 270   RETURN J
 280 FNEND
>TRACE Q,X(*)
>RUN
@20 Q=5
@30 X[1]=2.5
@30 X[2]=75.6
@30 X[3]=36.2
@30 X[4]=15.7
@30 X[5]=100
 5
```

The command TRACE Q,X(*) traces the variable Q and prints its value at line 20. This is the only value printed for Q since its value is not subsequently changed. Then the five values of the numeric array X(*) are printed with the line at which they assume these values.

The program output in this case is 5, the result returned by the function FND.

This trace is turned off by:

```
>UNTRACE
```

UNTRACE without a trace element list turns off all current traces.

In the trace below of the function FND and its local variables A(*) and N, the only trace output is for the function FND; its local variables are not traced since their values are unchanged by program execution.

```
>TRACE FND,FND(A(*),N)
>RUN
@FND
@270 FND(FND)=5
 5
```

A trace of statement execution is printed when a label or range of labels are used as parameters:

```
>TRACE 5Ø,2ØØ-28Ø
>RUN
*TRACE  5Ø
@FND
*TRACE  22Ø
*TRACE  23Ø
*TRACE  24Ø
*TRACE  25Ø
*TRACE  26Ø
*TRACE  25Ø
*TRACE  26Ø
*TRACE  25Ø
*TRACE  26Ø
*TRACE  25Ø
*TRACE  26Ø
*TRACE  27Ø
@27Ø FND(FND)=5
 5
*TRACE  21Ø
```

Because of the FOR statement in line 240, lines 250 and 260 are repeated 4 times. Following execution of line 270, control passes to line 50 where the value returned by the function is printed. The program then reaches line 210 and halts.

The command TRACE with no list prints the current trace element list.

```
>TRACE
FND(A[* ],N),5Ø,2ØØ-28Ø.
```

Following the UNTRACE command, the TRACE command has nothing to list.

```
>UNTRACE
>TRACE
>
```

The use of TRACE PROG is illustrated by the following segmented programs. ALPHA1 calls program BETA1 which in turn calls GAMMA1. Because the INVOKE statement is used, control returns to ALPHA1 (see Section X, Segmentation).

```
GAMMA1
  10 REM      PROGRAM GAMMA1
  20 PRINT "IN GAMMA1  -- RETURN TO BETA1"


BETA1
  10 REM      PROGRAM BETA1
  20 INVOKE "GAMMA1"
  25 PRINT "BACK IN BETA1  --  RETURN TO ALPHA1"


ALPHA1
  10 REM      PROGRAM ALPHA1
  20 INVOKE "BETA1"
  25 PRINT "BACK IN ALPHA1  -- TERMINATE"


>TRACE PROG
>RUN
ALPHA1
*TRACE, INVOKE: BETA1
*TRACE, INVOKE: GAMMA1
IN GAMMA1  -- RETURN TO BETA1
*TRACE, REVERT: BETA1
BACK IN BETA1  --  RETURN TO ALPHA1
*TRACE, REVERT: ALPHA1
BACK IN ALPHA1  -- TERMINATE
```

BREAK/UNBREAK Commands

The BREAK command allows the user to specify points where the execution of a program should be interrupted (or "broken"). A break point is a label, a range of labels, or any point at which a transfer is made from one program to another (through CHAIN, INVOKE, or END). UNBREAK turns off break points.

Forms

The forms of the commands are:

BREAK

BREAK breakpoint list

UNBREAK

UNBREAK breakpoint list

The items in the optional *breakpoint list* include: *label*, *label-label*, and PROG. They are specified in the same way as for TRACE.

Explanation

BREAK or UNBREAK can be specified before the program is run or when it is broken. When the program is run, execution suspends just before execution of a statement whose label is in the breakpoint list. If PROG is specified, execution suspends after a program is brought into the user's work area by CHAIN, INVOKE, or END but before the program is run (see Section X, Segmentation).

When execution suspends as a result of a breakpoint, the statement label about to be executed is printed in the form:

*BREAK *label*

*BREAK *programname label*

The *programname* is listed only if the program has been named.

After this output, a > is printed to indicate that a command can be entered. The legal commands during a break period are listed below.

Execution is resumed with the RESUME or GO command.

BREAK with no parameters causes all the current breakpoints to be listed.

UNBREAK with no parameters deletes all current breakpoints. With parameters, UNBREAK deletes those breakpoints specified by the labels in the breakpoint list.

LEGAL COMMANDS DURING BREAK

Certain commands may be used only during a break period:

ABORT

CALLS

FILES

GO

RESUME

SET

SHOW

These commands are described in this section.

Of the remaining commands, only these are legal during a break period:

BREAK/UNBREAK

CATALOG

CREATE

DUMP

EXIT

KEY

LENGTH

LIST

SPOOL

SYSTEM

TRACE/UNTRACE

XEQ

If the user enters any other command, BASIC/3000 responds:

```
ILLEGAL WHILE RUN SUSPENDED.  DO YOU WANT TO ABORT?
```

The user enters anything starting with "Y" to abort the current program and carry out the command, or enters anything else not to abort the program and to ignore the command.

During a break period, the user can type ABORT to terminate his current run and return to BASIC command mode where all commands are legal.

The commands SCRATCH and GET (illegal during a break period) will clear all traces and breakpoints. RUN *programname* will also clear traces and breakpoints, but a RUN without the *programname* parameter will not.

The CHAIN and INVOKE commands clear traces and breakpoints except when PROG is used. INVOKE saves traces and breakpoints and restores them upon return to the invoking program.

Examples

```
 10 DIM A[5,10]
 20 MAT READ A
 30 DATA 10,20,30,40,50,100,200,300,400,500
 40 DATA 110,120,130,140,150,210,220,230,240,250
 50 DATA 310,320,330,340,350,410,420,430,440,450
 60 DATA 510,520,530,540,550,610,620,630,640,650
 70 DATA 710,720,730,740,750,810,820,830,840,850
 80 RESTORE 50
 90 READ (FOR X=1 TO 3,(FOR Y=1 TO 10,A[X,Y]))
100 END
>BREAK 30,100
>RUN
*BREAK  30
>SHOW A(1,1),A(5,10)
A[1,1]=10
A[5,10]=850
>GO
*BREAK  100
>SHOW A(1,1),A(5,10)
A[1,1]=310
A[5,10]=850
>UNBREAK
>ABORT
```

During the breakpoints, the command SHOW (described later in this section) causes the values of the specified elements to be printed. After the program has run with two breakpoints at line 30 and 100, the breakpoints are deleted with UNBREAK. ABORT is used to return the user to the BASIC command mode. He may then run the program without breakpoints.

In the example below, the same program is named BRK1 and then run with the same breakpoints. The breakpoints are listed with BREAK during the second breakpoint and then deleted individually. GO finishes execution of the program after which the user is returned to BASIC command mode and runs the program without breakpoints:

```
 10 DIM A[5,10]
 20 MAT READ A
 30 DATA 10,20,30,40,50,100,200,300,400,500
 40 DATA 110,120,130,140,150,210,220,230,240,250
 50 DATA 310,320,330,340,350,410,420,430,440,450
 60 DATA 510,520,530,540,550,610,620,630,640,650
 70 DATA 710,720,730,740,750,810,820,830,840,850
 80 RESTORE 50
 90 READ (FOR X=1 TO 3,(FOR Y=1 TO 10,A[X,Y]))
100 END


>NAME BRK1
>BREAK 30,100
>RUN
BRK1
*BREAK BRK1 30
>SHOW A(1,1)
A[1,1]=10
>GO
*BREAK BRK1 100
>SHOW A(1,1)
A[1,1]=310
>UNBREAK 100
>BREAK
30.
>UNBREAK
>GO

>RUN
BRK1
```

ABORT Command

The ABORT command is legal only during a break period; it terminates the suspended program and returns the user to a normal state where all commands are legal.

Form

ABORT

Explanation

When ABORT is specified, the break period is ended and the run terminated. The user can now enter any command legal during normal BASIC execution, but cannot enter the commands legal only during a break period.

Examples

```
  10 DIM A[5,10]
  20 MAT READ A
  30 DATA 10,20,30,40,50,100,200,300,400,500
  40 DATA 110,120,130,140,150,210,220,230,240,250
  50 DATA 310,320,330,340,350,410,420,430,440,450
  60 DATA 510,520,530,540,550,610,620,630,640,650
  70 DATA 710,720,730,740,750,810,820,830,840,850
  80 RESTORE 50
  90 READ (FOR X=1 TO 3,(FOR Y=1 TO 10,A[X,Y]))
 100 END
>BREAK 30,100
>RUN
*BREAK  30
>SHOW A(5,10)
A[5,10]=850
>ABORT
```

RESUME or GO Command

The RESUME command ends the interactive debugging mode and resumes the suspended program. This command is legal only during a break period. GO may be used instead of RESUME; there is no difference between them.

Form

RESUME

RESUME label

GO

GO label

Explanation

A RESUME by itself restarts the program at the location printed when the program break occurred. A RESUME with a label restarts execution at that location, unless that location transfers into or out of a function from the current location.

The label parameter for RESUME or GO is not allowed when the break occurs as a result of pressing *CTRL Y*. RESUME or GO without a label may be used to resume suspended operation as a result of a *CTRL Y* break.

Examples

Using the same program, a breakpoint is specified for line 30 where the array A is displayed. During this break another breakpoint is specified for line 100 and then the suspended program is resumed at line 80 (RESUME 80). At the next breakpoint, array A is again displayed. GO is typed after the final breakpoint to complete execution of the program:

```
 10  DIM A[5,10]
 20  MAT READ A
 30  DATA 10,20,30,40,50,100,200,300,400,500
 40  DATA 110,120,130,140,150,210,220,230,240,250
 50  DATA 310,320,330,340,350,410,420,430,440,450
 60  DATA 510,520,530,540,550,610,620,630,640,650
 70  DATA 710,720,730,740,750,810,820,830,840,850
 80  RESTORE 50
 90  READ (FOR X=1 TO 3,(FOR Y=1 TO 10,A[X,Y]))
100  END

>BREAK 30
>RUN
*BREAK  30
>SHOW A(1,1),A(3,10),A(5,10)
A[1,1]=10
A[3,10]=450
A[5,10]=850
>BREAK 100
>RESUME 80
*BREAK  100
>SHOW A(1,1),A(3,10),A(5,10)
A[1,1]=310
A[3,10]=850
A[5,10]=850
>GO

>
```

Note that GO 80 could have been used instead of RESUME 80, and RESUME instead of the final GO with no effective change to this example.

SHOW Command

The SHOW command prints the values of the items specified; this command is legal only during a break period.

Form

The form of SHOW is

SHOW item list

The list can include:

- variables (numeric or string)
- array elements
- entire arrays (*name* (*) for one-dimensional array or *name* (*,*) for two-dimensional array)
- local variables (*function name* (*variable list*))

Explanation

An array is printed as in the MAT PRINT statement (see Section III), except that undefined values are noted with the word UNDEFINED. The variable list in parentheses that follows a function name can include only local variables of that function. The function must be active; that is, the function must have been called and not be completed.

Examples

The example below specifies breakpoints for line 20 and lines 70 through 90. Since line 80 is not executed, breaks actually occur in lines 70 and 90. SHOW commands are used to print the contents of the variable X$ at the break in line 70 and the contents of the local variable A$ in function FNR$ at the break in line 20. At the break in line 90, an attempt is made to show the contents of the non-existent array and of an existing array that has not been given any values. A new breakpoint at line 100 is specified where the SHOW command is used to print the previously undefined array B. GO continues execution of the program until it ends.

```
   10 DEF FNR$(A$)
   20   IF LEN(A$)<=1 THEN RETURN A$
   30   RETURN FNR$(A$[2])+A$[1,1]
   40 FNEND
   50 DIM X$[5],B[2,5]
   60 X$="12345"
   70 IF FNR$(X$)="54321" THEN PRINT "YES"
   80 ELSE PRINT X$
   90 MAT READ B
  100 DATA 10,20,30,40,50,60,70,80,90,100
  110 END
>BREAK 20,70-90
>RUN
*BREAK  70
>SHOW X$
X$="12345"
>GO
*BREAK  20
>SHOW FNR$(A$)
FNR$:A$="12345"
>UNBREAK 20
>GO
YES
*BREAK  90
>SHOW A(*)
A DOES NOT EXIST
>SHOW B(*,*)
B[*]
UNDEFINED   UNDEFINED   UNDEFINED   UNDEFINED   UNDEFINED

UNDEFINED   UNDEFINED   UNDEFINED   UNDEFINED   UNDEFINED

>BREAK 100
>GO
*BREAK  100
>SHOW B(*,*)
B[*]
 10     20     30     40     50

 60     70     80     90     100

>GO
```

SET Command

The SET command allows the user to set any variable to a constant value; this command is legal only during a break period.

Form

The form of the SET command is

SET item = constant

The items to be set can include variables and array elements and local variables, specified as in the SHOW command, except that the form using asterisks may not be used.

Examples

```
  10 DIM X[5]
  20 MAT READ X
  30 DATA 273.1,765.3,795.1,654.9,195.7
  40 PRINT FND(X[* ],5)
  50 END
 210 DEF FND(A[* ],INTEGER N)
 220   REAL I,J
 230   J=1
 240   FOR I=2 TO N
 250     IF A[I]>A[J] THEN J=I
 260   NEXT I
 270   RETURN J
 280 FNEND
>BREAK 30
>RUN
*BREAK  30
>SHOW X(1)
X[1]=273.1
>SET X(1)=950.2
>SHOW X(1)
X[1]=950.2
>GO
 1
```

The result of the program is changed by setting the first element in the array X to a higher value than the other elements.

When the break points are removed, the program runs with the data read from the DATA statement in line 30:

```
>UNBREAK
>RUN
 3
```

FILES Command

The FILES command is legal only during a break period. It prints a list of all the files that are currently open in the executing program. The list is by name and internal file number (see Section VIII, Files).

Form

FILES

Explanation

When FILES is typed during a break, a list of the file numbers specified by the FILES statement in the executing program is printed. The numbers are in ascending order and each is followed by a file name if the file is open, by an asterisk if the file number is reserved but not yet open, or by *#n* where *n* is the file number of a file opened in another program that called the current program with INVOKE. The file name of an open file is qualified by the group name and account name.

Examples

In the first example, FILES specified in the break at line 30 shows four open files. The break at line 40, after the ASSIGN statement closed file number 5, shows only three files currently open:

```
  10 REM    PROGRAM ONE
  20 FILES A,B,*,C,D
  30 ASSIGN *,5
  40 END
>BREAK 30
>RUN
*BREAK  30
>FILES
1  A.BASIC.LANG
2  B.BASIC.LANG
3  *
4  C.BASIC.LANG
5  D.BASIC.LANG
>BREAK 40
>GO
*BREAK  40
>FILES
1  A.BASIC.LANG
2  B.BASIC.LANG
3  *
4  C.BASIC.LANG
5  *
```

In this example, program FIRST calls program SECOND with INVOKE. BREAK PROG is used to specify a breakpoint when control goes to SECOND and again when control reverts to FIRST. In SECOND, three local files are open, one of which is internal file #2 or the file B. It also shows the internal files A and B that were opened in FIRST and remain open following the INVOKE. The FILES command at the break upon return to FIRST shows that only the two files local to FIRST are open and that file #3 has been reserved:

```
SECOND
  10 REM     PROGRAM SECOND
  20 FILES C,#2,D
  30 END

FIRST
  10 REM     PROGRAM FIRST
  20 FILES A,B,*
  30 INVOKE "SECOND"
>BREAK PROG
>RUN
FIRST
*BREAK, INVOKE: SECOND
>FILES
1  A.BASIC.LANG
2  B.BASIC.LANG
3  *
4  C.BASIC.LANG
5  #2
6  D.BASIC.LANG
LOCAL FILES START AT 4
>GO
*BREAK, REVERT: FIRST
>FILES
1  A.BASIC.LANG
2  B.BASIC.LANG
3  *
>GO

>
```

CALLS Command

The CALLS command is legal only during a break period. It prints a list of all functions that have not been completed, and of all programs that have been called with INVOKE but have not been completed by END. This list is in reverse chronological order starting with the most recent.

Forms

CALLS

Examples

At the breakpoint for statement 40, the CALLS command shows that function FNN called FNM. Note that functions are listed in reverse chronological order:

```
PROG1
  10 DEF FNM(A,B)=SGN(A)*FNN(ABS(A),ABS(B))
  20 DEF FNN(A,B)
  30   X=A-INT(A/B)*B
  40   RETURN X
  50 FNEND
  60 PRINT FNM(-4,3)
>BREAK 40
>RUN
PROG1
*BREAK PROG1 40
>CALLS
FNN
FNM
```

In the following example, ALPHA2 uses INVOKE to call BETA2; BETA2 uses CHAIN to call GAMMA2. Because GAMMA2 returns to ALPHA2, not BETA2, a CALLS command entered during the break in GAMMA2 shows that ALPHA2 invoked GAMMA2:

```
GAMMA2
  10 REM     PROGRAM GAMMA2
  20 PRINT "IN GAMMA2 -- RETURN TO ALPHA2"

BETA2
  10 REM      PROGRAM BETA2
  20 CHAIN "GAMMA2"



ALPHA2
  10 REM      PROGRAM ALPHA2
  20 INVOKE "BETA2"
  25 PRINT "BACK IN ALPHA2 -- TERMINATE"
>BREAK PROG
>RUN
ALPHA2
*BREAK, INVOKE: BETA2
>GO
*BREAK, CHAIN: GAMMA2
>CALLS
INVOKED BY ALPHA2
```

Each of the following three programs contains at least one function definition and function call. Function FNA in program ALEF1 calls FNB wherein ALEF1 calls BET1 with an INVOKE statement. At the breakpoint in line 40 of GIMEL, function FNE calls FNF. The CALLS command entered during the breakpoint in GIMEL shows a complete history of all nested function calls and INVOKE statements in reverse chronological order:

```
GIMEL
  10 REM      PROGRAM GIMEL
  20 DEF FNE(X)=FNF(X)
  30 DEF FNF(X)
  40   PRINT "IN GIMEL"
  50   RETURN 0
  60 FNEND
  70 X=FNE(4)

BET1
  10 REM      PROGRAM BET1
  20 DEF FNC(X)
  30   INVOKE "GIMEL"
  40   RETURN 0
  60 FNEND
  70 X=FNC(3)


ALEF1
  10 REM      PROGRAM ALEF1
  20 DEF FNA(X)=FNB(X)
  30 DEF FNB(X)
  40   INVOKE "BET1"
  50   RETURN 0
  60 FNEND
  70 X=FNA(2)
>BREAK PROG
>RUN
ALEF1
*BREAK, INVOKE: BET1
>GO
*BREAK, INVOKE: GIMEL
>BREAK 40
>GO
*BREAK GIMEL 40
>CALLS
FNF
FNE
INVOKED BY BET1
FNC
INVOKED BY ALEF1
FNB
FNA
```

WAIT Command

The WAIT command suspends the BASIC Interpreter with the PAUSE intrinsic for a specified period of time. The command has the following form:

```
WAIT time1 [,time2]
```

where "time1" and "time2" have the following form:

```
[[hours:] minutes:] seconds
```

"seconds" may be a floating-point value; it must be less than 60 if "minutes" is present. "Minutes" and "hours" must be integral values. "Minutes" must be less than 60 if "hours" is specified.

If only "time1" is specified, the Interpreter suspends for the indicated period of time. If "time2" is also specified, the Interpreter suspends for a random time period between "time1" and "time2". This command is useful for scheduling events during a benchmark.

WARNING: This command cannot be terminated with control-Y.