Xitan, Inc.
Disk BASIC Version 1.06

Preliminary Update Documentation

Written by Neil J. Colvin
June 16, 1978

Xitan Disk Basic Version 1.06
Preliminary Update Documentation

1. Program Text Inputting and Editing

A number of changes and additions have been made to the facilities provided for the manipulation of the program text.

1-1. Keyboard Input

The BACKSPACE key is now recognized as a character delete in addition to the DEL key. The key echos as BS-SPACE-BS. This facilitates keyboard input on a video device which allows backspacing and overwriting.

1-2. AUTO Command

The AUTO command has been enhanced to provide better user control and flexibility. The format of the command is:

```
     AUTO [<starting line>[+]][,<increment>]
```

The only change in this format is the + option. The use of the + option specifies that the increment is to be added to the starting line before the first line number is generated.

If any line number generated by the AUTO command corresponds to an already existing line in the program, the line number is preceeded by a "+" when displayed on the terminal. To avoid overwriting the existing line, the entry of JUST the RETURN key will advance the line number to the next line without changing the exisitng one.

This automatic skipping of line numbers while in AUTO mode may be used at any time. Previously, the entry of just the RETURN key terminated the AUTO mode. Because of this new feature, a new AUTO mode termination has been provided. To terminate AUTO mode, enter CTL-E. This will immediately return to command mode, regardless of any other text entered on the same line.

1-3. The "." Line Number

To facilitate program entry and editing, a shorthand notation for the "current line" is provided. From command mode (ONLY), a period (".") may be substituted anywhere a line number is normally used. This period represents the last program line accessed by a program editing or execution command. For example:

```
     10 A$=MID$(B$,5,6
```

Note that the EDIT command references line 10, the last line accessed.

After any execution error, the period is set to the line the error occured on, so that a

```
     LIST .
```

will list the line in error.

When AUTO mode is terminated by a CTL-E, the period is set to the last line entered. To resume input with the next line, the command:

```
     AUTO .+
```

may be used (see AUTO command, Section 1-2).

1-4. FIND Command

The FIND command allows those lines within the basic program which contain a specific text string to be easily located. The format of the command is:

```
     FIND <delimited text>[,<line range>]
```

where <delimited text> is the desired text string (32 characters maximum) delimited (preceeded and followed) by any character (except a comma, space, or tab) not contained within the string, and <line range> is a normal line number range specification. If <line range> is omitted, the entire program will be searched.

All lines within the specified range containing the text will be located and displayed on the terminal.

For example:

```
     FIND /COS(/, 100-300
```

will find all lines containing "COS(" between lines 100 and 300 inclusive.

1-5. REPLACE Command

The REPLACE command is an extension of the FIND command which not only locates the specified text string, but also replaces it with another string. The format of the command is:

```
     REPLACE <delimited text 1><delimited text 2>,<line range>
```

Note that the <line range> is NOT optional on this command.

This command will replace all occurences of <delimited text 1> within the specified range by <delimited text 2>. Each line in which a substitution occurs is displayed in its new form on the console.

If any replacement would cause the line to exceed the maximum size (254 characters) the command is aborted with a "String Too Long" error.

For example:

```
     REPLACE /COS(/ \SIN(\, 100-300
```

will replace all occurences of "COS(" between lines 100 and 300 inclusive with "SIN(". Note that the delimiter need not be the same for both strings.

1-6. LOAD Command

The LOAD command has been simplified. The ALOAD command no longer exists. The format of the LOAD command is:

where <file name> is a string value (constant, variable or expression) which contains a standard CP/M file name. If the CP/M file extension (type) is omitted, it defaults to ".BAS".

The specified disk file is located and examined to determine if it contains an ASCII or internal format BASIC program. The appropriate LOAD procedure is then automatically performed. (Note that the internal format of Disk BASIC Version 1.06 programs is significantly different from previous internal formats, and completely incompatible. To transfer from one to another, ASCII format disk files should be used.)

For example:

```
     LOAD "PROGRAM"
```

loads a program from the disk file "PROGRAM.BAS".

1-7. SAVE Command

The SAVE command has been simplified. The ASAVE command no longer exists. The format for the SAVE command is:

where <file name> is as in the LOAD command. The normal SAVE command (without the "A" option) saves the current program on disk in the specified file in internal format. The use of the "A" option causes the program to be saved in ASCII rather than internal format.

For example:

```
     SAVE "PROGRAM",A
```

saves the current program on disk as "PROGRAM.BAS" in ASCII format.

1-8. RESAVE Command

The RESAVE command has been added to simplify the process of working with a single program. The format of the RESAVE command is:

```
     RESAVE <file name>[,A]
```

where the arguments are the same as in the SAVE command.

The operation of this command is identical to that of the SAVE command with one exception: the SAVE command gives an error if the specified file already exists on disk; the RESAVE command ERASEs the file if it already exists.

1-9. MERGE Command

The MERGE command replaces the AMERGE command. Its format is:

```
     MERGE <file name>
```

where <file name> is as in the LOAD and SAVE commands.

The file specified must be in ASCII format. It is merged, line by line, with the current program.

For example:

```
     MERGE "SUB1"
```

merges the program in the disk file "SUB1.BAS" with the current program in memory.

1-10. LOADGO Command

The only change in the LOADGO command is its format. The new format is:

```
     LOADGO <file name>[,<start line>]
```

where <file name> is as in the LOAD and SAVE commands.

The file specified by <file name> must contain a program in internal format or a "Syntax Error" will occur.

1-11. PRIVACY Statement

To provide for the security of the source for a BASIC program, the PRIVACY statement has been added. The format of the statement is:

```
     PRIVACY <password expression>
```

where <password expression> is any string value (constant, variable or expression).

When this statement is present in a BASIC program, the source text of the program may only be accessed and modified with the knowledge of the value of the <password expression>. The presence of this statement modifies the syntax of many of the text editing and inputting commands, requiring the prefacing of a password to the command arguments. The commands affected are: LIST, EDIT, DELETE, AUTO, SAVE, RESAVE, COPY, RENUMBER, FIND and REPLACE. In addition, no direct program statement entry or deletion (<line number> <text>) is allowed at all. The password is a string value which must be equal to the declared PRIVACY <password expression> value. It must be followed by a comma if more arguments are to follow. For example, if the source program were:

```
     10 A=5
     20 B=6
     30 D=A*SQR(B)
     40 PRINT D
     50 PRIVACY "SQUINT"
     60 END
```

then the following command would be required to list the entire program:

```
     LIST "SQUINT"
```

The PRIVACY statement may be anywhere in the program, but MUST be the first statement on the line.

2. Arithmetic and Logical Operators

The set of available arithmetic and logical operators has been expanded. The complete set, in priority order (the order in which they are evaluated) is as follows:

a. Expressions in parentheses "()"

b. ^ (exponentiation)

c. - (unary minus)

d. * and / (multiplication and division)

e. \ (integer division)

f. MOD (modulus)

g. + and - (addition and subtraction)

h. relational operators
   = (equal)
   <> (not equal)
   < (less than)
   > (greater than)
   <= and =< (less than or equal to)
   >= and => (greater than or equal to)

i. NOT (logical bitwise complement)

j. AND (logical bitwise and)

k. OR (logical bitwise or)

l. XOR (logical bitwise exclusive or)

m. EQV (logical bitwise equivalence)

n. IMP (logical bitwise implication)

All operators listed at the same level in the table are evaluated left to right in an expression.

All logical operations convert their operands to sixteen bit integer values prior to the operation. These operands must be in the range 0 to 65,535 or -32,768 to 32,767. An "Illegal Function Call" error will result if the operands are not within this range.

3. Intrinsic Functions

A number of new mathematical and string functions have been added.

3-1. Mathematical Functions

a. LOG10(X) : returns the base ten logrithm of X

b. FIX(X) : returns the truncated integer part of x

c. PI : [no argument] returns the value of pi

d. EE : [no argument] returns the value of e

e. RND : [no argument] returns a random number, same as RND(X) when X>0

f. TIME : [no argument] returns the time in milliseconds since midnight (only on systems with real time clock support, otherwise returns zero)

3-2. String Functions

a. HEX$(X) : returns a string containing the hexedecimal representation of X converted to a sixteen bit integer

b. SPACE$(X) : returns a string containing X spaces (X must be less than 256)

c. STRING$(S$,X) : returns a string containing the string S$ repeated X times (X*LEN(S$) must be less than 256)

d. FIX$(S$,X) : returns a string that is X characters long whose value is S$ either truncated or padded with spaces to the correct length (X must be less than 256)

e. DATE$: [no argument] returns a string whose value is the current date in the form MM/DD/YY (only on systems with real time clock support, otherwise returns a string of eight spaces)

f. TIME$: [no arguments] returns a string whose value is the current time in the form HH:MM:SS (only on systems with real time clock support, otherwise returns a string of eight spaces)

4. Input/Output Operations

The input/output operations in BASIC are significantly changed in this version. Prior to detailed descriptions of each of the various commands, statements, and functions, some basic concepts should be understood.

4-1. Unit Numbers

Because the BASIC now supports multiple I/O devices (console, list, reader, punch, disk), a method is provided to direct a particular I/O operation to a specific device. The mechanism for this is the unit number. Each I/O device, and each active file on the disk, is assigned a unique unit number from 0 to 255. The unit numbers associated with specific devices are fixed as follows:

```
0 : the console device
1 : the LOAD/SAVE device (normally disk)
2 : the list device
3 : the reader device
4 : the punch device
5-9 : reserved for future expansion
10-255 : the disk device
```

Note that these devices correspond to the standard CP/M supported devices.

All I/O operations (except LOAD/SAVE) may be directed to any I/O device which is capable of supporting that operation (eg. a PRINT cannot be done to the reader device). The default unit number for all I/O operations (except SAVE/LOAD) is 0 (the console).

The format for specifying a unit number is "#<unit>", where <unit> may be any expression evaluating to a valid unit number. In intrinsic functions which take unit numbers as arguments, the "#" is optional. If the unit number is to be followed by additional arguments, it must be followed by a comma.

For the disk device, any unit number 10 through 255 is valid. The actual association between a unit number and a specific disk file is made by the OPEN command described below.

4-2. Random Addresses

For the disk device, an expanded unit specification is allowed by many of the I/O operations. This specification includes not only the unit number (specifying a particular disk file), but also an optional random address within that disk file. This random address represents the particular record within the file at which the I/O operation will start (For more information on records, see the OPEN statement). The format of this expanded unit specification is:

```
     #<unit>[@<random address>]
```

where <random address> is any expression evaluating to a positive integer value less than 4194304.

If specified, the random address is multiplied by the record size to generate a "byte pointer". This byte pointer specifies the particular byte in the disk file at which the I/O operation will start (0 is the first byte in the file). If not specified, the I/O operation will normally proceed from the current byte position (the first byte not processed by the previous I/O operation). The exception to this is files OPENed in the Update mode (see the OPEN statement).

4-3. OPEN Statement

The OPEN statement initializes the I/O device for I/O operation. Each device (and associated unit numbers) has its own specific actions and format for the OPEN statement. The general format of the statement is:

```
     OPEN #<unit>,<mode>{,<file name>[,<record size>]}
```

where <unit> is as described in Section 4-1, <mode> is a string value (constant, variable or expression) which contains the single character I, O, R or U. The arguments in braces are used for disk units only, and will be described below.

The mode values are as follows:

I	Input mode. Only input operations may be done on the unit.
O	Output mode. Only output operations may be done on the unit.
R	Random mode. Both input and output operations may be done on the unit. Valid only for disk units.
U	Update mode. Both input and output operations may be done on the unit. Unless otherwise specified however, each output operation

input operation begins at the first unprocessed byte address from the previous I/O operation. Valid only for disk units.

4-3-1. Console (Unit 0)

The console unit is always open. An OPEN to the console unit simply causes a form feed (hex 0C) to be output to the device, and the unit parameters to be reinitialized to the defaults (see Unit Parameters).

4-3-2. List (Unit 2)

The list unit is always open. An OPEN to the list unit (must be mode 0) simply causes a form feed (hex 0C) to be output to the list device, and the device parameters to be reinitialized to their default values.

4-3-3. Reader (Unit 3)

The reader unit is always open. An OPEN to the reader unit simply causes the device parameters to be reinitialized to their default values.

4-3-4. Punch (Unit 4)

The punch unit is always open. An OPEN to the punch unit outputs sixteen bytes of leader (hex FF) to the device and reinitializes the device parameters to their default values.

4-3-5. Disk (Unit 10-255)

Disk units are dynamically allocated as requested by OPEN statements. The maximum number of disk units which may be OPENed simultaneously by the program is determined by the units parameter to the CLEAR statement (see the CLEAR statement below).

Each disk unit is associated with a specific disk file. This association is established by the <file name> argument in the OPEN statement. This <file name> is a string value which contains a standard CP/M disk file name. If omitted, the extension (type) is assumed to be ".BAS".

The way in which the association is made is determined by the OPEN <mode> as follows:

I The file is searched for on disk, and if found, the association is made. If not found, an "Input File Not Found" error is given.

where <unit> is an OPENed unit number. If one or more units are specified in the CLOSE, just those units will be closed. If no units are specified, ALL disk units which are open will closed. The specific actions taken for each type of unit are described below.

4-4-1. Console (Unit 0)

A CLOSE to the console unit only causes the output of a form feed to the console device. No other action takes place, and the unit remains open.

4-4-2. List (Unit 2)

A CLOSE to the list unit only causes the output of a form feed to the list device. No other action takes place, and the unit remains open.

4-4-3. Reader (Unit 3)

A CLOSE to the reader unit has no effect. The unit remains open.

4-4-4. Punch (Unit 4)

A CLOSE to the punch unit causes a CTL-Z (hex 1A) followed by sixteen bytes of leader (hex FF) to be output to the punch device. No other action takes place, and the unit remains open.

4-4-5. Disk (Unit 10-255)

A CLOSE to a disk unit causes different actions depending on the mode in which the unit is open, as follows:

I	No specific action takes place.

4-5. Device Parameters

Each I/O unit has associated with it a number of modifiable parameters:

line width : the number of characters output to a line on that unit before a carriage return/line feed is sent automatically

null count and character : the number and value of the characters sent to the device after each carriage return/line feed sequence for timing purposes

quote mode : the character (if any) output to delimit string values when outputting in ASCII mode

prompt character : the character (if any) output to prompt the user that an INPUT statement has been executed

Each of these parameters has a default value for each unit type, and is overidable by the use of the OPTION statement. The format of this statement is:

```
OPTION [#<unit>,]<option>[,<arg 1>[,<arg 2>]]
```

where <unit> is as above, <option> is a string value containing either W, N, Q or P. These options take different arguments as follows:

W width : <arg 1> is the width of the line desired (20-253)

N null : <arg 1> is the number of characters to output (0-255) and <arg 2> is the characters decimal value (defaults to 0 if omitted)

Q quote : <arg 1> is the decimal value of the character to be used as the outputted string delimiter (0 means NO delimiter, 34 is a quote mark)

P prompt : <arg 1> is the decimal value of the character to be output as the INPUT statement prompt (0 means no automatic prompt, 63 is a question mark)

Each unit has default parameters as follows:

```
     2 (list) : W[72],N[3,0],Q[0]

     3 (reader) : not applicable

     4 (punch) : W[253],N[0,0],Q[34]

     10-255 (disk) : W[253],N[0,0],Q[34]
```

Note that the OPTION statement replaces the NULL, WIDTH and QUOTE statements of previous versions.

4-6. Dynamic Unit Space

Each disk unit (10-255) requires 181 bytes of memory during the time that it is OPENed. When BASIC is started, the default is to allocate space for no disk units. To change this default allocation, the CLEAR statement is used. The format of the new CLEAR statement is:

```
     CLEAR [<string space>][,<number of units>]
```

where <string space> is the amount if string area to allocate, and <number of units> is the number of simultaneously OPENed disk units to allocate. If either argument is omitted, the corresponding allocation remains unchanged. The use of the CLEAR statement implicitely results in the disassociation of all OPENed disk units from their corresponding disk files WITHOUT any CLOSE actions being taken.

Note that space is always allocated for the LOAD/SAVE unit (which is effectively a disk unit).

4-7. Data Input and Output

The format of the data input and output commands is unchanged except for the addition of the extended unit specifier option for disk units. The formats are:

```
     INPUT [LINE] [#<unit>[@<addr>],] [<prompt>;] <i/o list>

     PRINT [#<unit>[@<addr>],] [USING <format>;] [<i/o list]

     READ #<unit>[@<addr>],<i/o list>

     WRITE #<unit>[@<addr>],<i/o list>

     MAT READ #<unit>[@<addr>],<i/o list>

     MAT WRITE #<unit>[@<addr>],<i/o list>
```

Note the reversal of INPUT LINE from LINE INPUT in previous versions. Also, MSAVE has become MAT WRITE and MLOAD has become MAT READ. The remainder of the statements in each of these cases is unchanged.

It is also important to note that binary I/O (READ and WRITE) operations can only be done to binary devices (not the console [unit 0]).

4-7-1. OUTBYTE Statement

The OUTBYTE statement has been added to facilitate single byte output operations to defined units. The format of the statement is:

```
     OUTBYTE [#<unit>[@<addr>],] <i/o list>
```

If the <i/o list> element is a numeric value, it must be 0-255, and is output to the specified unit as a single eight-bit byte. If the element is a string, each character of the string is output as a single byte, with no formatting of any sort. The length of the string is not output as it is with a WRITE statement.

4-7-2. SETLOC Statement

The SETLOC statement is provided to set the byte address for a disk unit, independent of any I/O operation. This allows the random address to be determined one place in the program, and all I/O to be done sequentially in another place. The format of the statement is:

```
     SETLOC #<unit 1>@<addr 1> [,#<unit 2>@<addr 2> ...]
```

where each unit's byte address is set as specified.

4-7-3. ON EOF Statement

The ON EOF statement has been expanded to allow a seperate EOF statement for each unit currently opened. The format is:

```
     ON EOF [#<unit>] [GOTO [<line number>]]
```

where <line number> is where execution should continue when an End-of-File is encountered.

If the <unit> is specified, than the EOF branch applies only to that unit. If no unit is sepcified, than the EOF branch applies to all units with no EOF branches specified. If the

An EOF branch may be set for the console unit (0), and will be taken whenever a CTL-Z (hex 1A) is received from the console.

The ON EOF may be executed any number of times, and changed as desired.

4-7-4. EOF Statement

In a similar fashion to the above, the EOF statement may cause a software EOF trap on a specific unit. The new format is:

```
EOF [#<unit>]
```

If executed, the effect is that of an EOF being encountered on the specified unit.

4-7-5. Intrinsic Functions

A number of new intrinsic functions have been added to increase the flexibility of the I/O system within BASIC. These are as follows:

a. POS(<unit>) : returns the number of characters output to the current line of the specified unit (counted in ASCII mode output only)

b. ERR(<unit>) : returns a logical TRUE (-1) is an I/O error was encountered during the last I/O operation on the specified unit (currently always FALSE [0])

c. EOF(<unit>) : returns a logical TRUE (-1) if an EOF was encountered during the last I/O operation on the specified unit (reset to FALSE [0] at the start of every I/O operation)

d. LOC(<unit>) : returns the byte address of the specified disk unit, the NEXT byte to be sequentially processed (this is always independent of any specified record size)

e. LOF(<unit>) : returns the number of bytes in the current extant of the disk file associated with the specified disk unit

f. BYTEPOLL(<unit>) : return a logical TRUE (-1) if a byte is available from the specified unit (the only time this will be FALSE [0] is for the console unit [0] when no character has been entered since the last input operation)

g. BYTE(<unit>) : returns the decimal value of the next byte read sequentially from the specified unit

h. BYTE$(<unit>) : returns a string of length one containing the next byte read sequentially from the specified unit

In each of these functions, the <unit> specified must be OPEN.

4-8. Expanded Capabilities for Other I/O

Other commands which perform output have also been enhanced to utilize the new extended unit specification. The formats of these enhanced commands are:

```
     LIST         [<password>,]         [#<unit>[@<addr>],]
              [<line number range>]

     LVAR [#<unit>[@<addr>]]

     TRACE [#<unit>[@<addr>],] <logical value>
```

Note that the previously provided "L" forms of these commands are no longer available (LLIST, LTRACE, LLVAR).

5. Disk File Management

The ERASE and RENAME commands are unchanged, as is the LOOKUP function, with the following formats:

```
     ERASE <file name>

     RENAME <file name 1>,<file name 2>

     LOOKUP(<file name>)
```

In addition, three new file management statements have been added.

5-1. DIR Command

The DIR command allows the display of the disk directory from within BASIC. The format of the command is:

```
     DIR [#<unit>[@<addr>],] [<file name>]
```

where <file name> is a string value (constant, variable or expression) containing a valid CP/M disk file name (with ? and * masking if desired). If omitted, the name defaults to "*.BAS", and if the extension is omitted, it defaults to ".BAS". All the files matching the specification are output to the specified unit.

For example, the following command outputs a list of all of the files on disk A to the console:

```
     DIR "A:*.*"
```

5-2. PROTECT Statement

The PROTECT command is only applicable to those systems which have individual disk file protection, on all others it does nothing. The format of the command is:

```
     PROTECT <file name>,<protection>
```

where <file name> is a string value containing a (possibly masked) CP/M file name, and <protection> is an integer value between 0 and 7 specifying the new protection key.

5-3. RESET Statement

Under CP/M, the changing of a diskette while BASIC (or any program) is running requires updating the operating systems tables. The RESET statement causes that to happen. The format is:

RESET

This command should not be issued with any non-Input mode files OPEN.

6. Program Controlled Console I/O

One of the new I/O functions provided in this version of BASIC is BYTEPOLL. This function determines if a byte of data is ready to be read from an I/O unit. The only unit this really applies to is the console. However, the usefullnes of this function is limited by the fact that BASIC itself constantly tests (and reads) the console input to determine if a CTL-E or other control function has been entered. Hence, under normal conditions, BYTEPOLL will never return a TRUE from the console.

To allow this prgrammed control of console input to work properly, a special statement has been added to the BASIC, the INTERRUPT statement. The format of this statement is:

```
INTERRUPT <interrupt logical>
```

The function of this statement is to set the internal console interrupt test to the value of the <interrupt logical>. If that value is TRUE (-1), then BASIC will continue (or resume) testing for CTL-E and other control functions. If it is FALSE (0), then the internal testing will stop. THIS MEANS THAT THE PROGRAM MAY NO LONGER BE STOPPED BY CTL-E. If a program logic error occurs in this mode, and some form of loop takes place, the only method for stopping the program will be resetting the processor.

INTERRUPT is automatically set to TRUE whenver BASIC returns to command mode.

7. Program Execution Control Statements

Two new execution control statements have been added to increase program control over the execution environment. In addition, to avoid a keyword conflict, a new statement has been added.

7-1. Return to Operating System

To leave BASIC and return to the operating system, the BYE command is used. The format of the command is:

```
     BYE
```

The execution of the BYE command is TERMINAL. Make sure that all output files are CLOSEd prior to issuing this command, or data may be lost.

This command replaces the EXIT command of previous versions.

7-2. RETURN Statement

The RETURN statement has been enhanced to provide a "non-standard" subroutine return capability. The format of the RETURN command is:

```
     RETURN [<line number>]
```

or

```
     ON <8-bit value> RETURN <line 1>[,<line 2>,...]
```

where <line number> is the line to RETURN to.

If no line number is specified, the operation of the RETURN command is unchanged. Specifying a line number causes the RETURN to terminate the corresponding GOSUB (as would normally happen), and then continue execution at the specified statement, NOT the statement following the GOSUB.

```
     10 GOSUB 40
     20 PRINT "LINE 20"
     30 STOP
     40 PRINT "LINE 40"
     50 RETURN 70
     60 STOP
     70 PRINT "LINE 70"
     80 END
```

The RUNning of this program would result in the output:

```
LINE 40
LINE 70
```

The ON ... RETURN format is provided for those cases where the non-standard RETURN is to one of a set of lines depending on some value.

7-3. EXIT Statement

The EXIT statement has been added to allow the correct early termination of a FOR-NEXT loop. The format of the EXIT statement is:

```
EXIT [<line number>][,][<variable name>]
```

where <line number> is the line to EXIT to, and <variable name> is the variable controlling the outermost FOR-NEXT loop to be terminated. Note that the comma is required only if both optional arguments are present.

Due to the fact that the BASIC interpreter allows FOR-NEXT loops to be structured in any fashion (including having the NEXT preceeding the FOR), a mechanism must be provided to specify the point at which the loop is to be considered terminated (as opposed to the normal completion of the loop at the NEXT statement). The EXIT statement provides this capability.

The EXIT statement with no arguments simply terminates the innermost currently active FOR-NEXT loop, leaving the controlling variable with its current value. The addition of a line number causes execution to continue at the specified line after the loop is terminated. This mode of operation is an exact replacement for a GOTO statement in the same context, and should be used whenever it is desired to jump out of a FOR-NEXT loop.

If more than one FOR-NEXT loop is currently active, and it is desired that other than the innermost one be terminated, the variable name for the controlling FOR-NEXT may be specified. In this case, all nested loops within the specified loop are also terminated. Note that the line number may be optionally specified along with a variable.

For example:

```
10 FOR I=1 TO 10
20 IF MX(I)=0 THEN EXIT 60
30 NEXT I
40 PRINT "NO ROOM"
```

```
50 I=0
60 FOR J=1 TO 10
70 FOR I=J TO 10
80 MX(I)=MX(J)
90 IF MX(I)=0 THEN EXIT 140,J
100 NEXT I
110 NEXT J
120 PRINT "DONE"
130 STOP
140 ...
```

illustrates the proper use of the EXIT statement. If a GOTO were used in line 20 rather than an EXIT, a "NEXT without FOR" error would result at line 110 because the second FOR I statement would establish a new active loop at the same level as the previous FOR I, losing the FOR J entirely. With the EXIT statements added, the program works properly.