ERASE

Format ERASE <list of array names>

Purpose The ERASE statement cancels array definitions made with the DIM statement.

Remarks The ERASE statement erases specified arrays from memory, allowing the arrays to be redimensioned or freeing that memory for other purposes. An "Illegal function call" error will result if an attempt is made to erase a non-existent array.

See also DIM

Example

```
10 DIM X(15)
20 FOR I=1 TO 15
30 X(I)=I
40 NEXT I
50 FOR I=1 TO 15
60 PRINT X(I);
70 NEXT I
80 PRINT
90 ERASE X
100 DIM X(15)
110 FOR I=1 TO 15
120 PRINT X(I);
130 NEXT I
140 END
```

```
RUN
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Ok
```


ClibPDF - www.fastio.com

ERROR

Format ERROR <integer expression>

Purpose The ERROR statement makes it possible to simulate occurrence of MFBASIC errors or to define other types of errors.

Remarks When this statement is executed in a program and no error trap routine is provided, the error message whose error code equals the value of <integer expression> is displayed and program execution halts; if the value of <integer expression> does not correspond to any error code which is defined in MFBASIC, an "Unprintable error" message is displayed. (A list of the MFBASIC error codes is given in Appendix A.) In any event, the value of <integer expression> must be greater than 0 and less than 255.

Error trap routines can be used in conjunction with the ERROR statement to assign user definitions to error codes. An example of this is shown in program example 2 below.

See also ON ERROR GOTO, RESUME, ERR/ERL, Appendix A

Example 1

```
10 INPUT A,B
20 X=A*B
30 IF X>50 THEN ERROR 6
40 GOTO 10
```

```
RUN
? 4,5
? 7,8
Overflow in 30
Ok
```


ClibPDF - www.fastio.com

Example 2

```
10 ON ERROR GOTO 80
20 CLS
30 INPUT "INPUT NUMBER FROM 1 TO 9";A
40 IF A<1 OR A>9 THEN ERROR 200
50 PRINT A
60 ERROR 210
70 END
80 IF ERR=200 THEN PRINT "ERROR":RESUME 30
90 IF ERL=60 THEN PRINT "TEST":RESUME 70
```

```
INPUT NUMBER FROM 1 TO 9? 0
ERROR
INPUT NUMBER FROM 1 TO 9? 10
ERROR
INPUT NUMBER FROM 1 TO 9? 7
 7
TEST
Ok
```


ClibPDF - www.fastio.com

FIELD

Format FIELD[#]<file number>, <field width> AS <string variable>, <field width> AS <string variable>,...

Purpose The FIELD statement is used to assign the positions in a random file buffer for use as variables.

Remarks When a file is OPENed, a buffer is automatically reserved in memory which is used for temporary storage of data while it is being transferred between a flexible disk (or other external device) and main memory. With random access files, data is read into this buffer from the flexible disk with the GET statement and written from the buffer to the disk with the PUT statement. However, before this can be done, a FIELD statement must have been executed to assign variables to specific positions in the random file buffer. (Disk data is then read into the variables in the buffer by executing a GET statement, or values are written into the variables in the buffer by executing a LSET or RSET statement. See Chapter 5 for detailed instructions on accessing random access files.) The <file number> which is assigned to this buffer is the number under which the file was OPENed. <field width> is the number of positions to be allocated to the specified <string variable>. For example:

```
FIELD 1,20 AS N$,10 AS ID$,40 AS ADD$
```

assigns the first 20 positions (bytes) of the buffer to string variable N$, the next 10 positions to ID$, and the next 40 positions to ADD$. The total number of positions assigned to variables by the FIELD statement cannot exceed the record length that was specified when the file was OPENed; otherwise, a "FIELD overflow" error will occur. (The default record length is 128 bytes.)
If necessary, any number of FIELD statements may be executed for the same file. If more than one such statement is executed, all assignments made are effective at the same time.

See also GET, LSET/RSET, OPEN, PUT

ClibPDF - www.fastio.com

Example

```
10 OPEN "R",#1,"TEST.DAT"
20 FIELD #1,8 AS A$,6 AS B$
30 RSET A$="EP"
40 LSET B$="SON"
50 PRINT A$+B$
60 PUT #1,1
70 CLOSE 1
80 END
```

```
RUN
    EPSON
Ok
```

NOTE:
Once a variable name has been FIELDed, use only RSET or LSET to store data in that variable. FIELDing a variable name assigns it to specific positions in the random file buffer; using an INPUT or LET statement to store values to FIELDed variable names will cancel this assignment and reassign the names to normal string space.

ClibPDF - www.fastio.com

FILES

Format FILES [<filename>]

Purpose The FILES command is used to display the names of files stored on the specified flexible disk.

Remarks If <filename> is omitted, the names of all files on the flexible disk in the currently active drive are displayed. <filename> may be represented using question marks (?) or asterisks (*). The question mark is a wildcard character which is used to indicate any character, while the asterisk indicates all characters in the specified position.

Example 1 FILES "L???????.BAS"
Displays the names of all files on the currently active disk whose primary names begin with the letter L and whose extensions are ".BAS".

Example 2 FILES "B:*.*" or FILES "B:"
Displays the names of all files on the disk in drive B.

Example 3 FILES "B:D???.*"
Displays the names of all files on the disk in drive B which begin with the letter D and include not more than four characters in the primary file name.

ClibPDF - www.fastio.com

FOR...NEXT

Format

FOR <variable> = <expression 1> TO <expression 2> [STEP <expression 3>]

.

.

NEXT [<variable>][,<variable>...]

Purpose

The FOR...NEXT statement makes it possible to repeat the series of instructions written between FOR and NEXT a specific number of times.

Remarks

With this statement, program execution loops through the series of instructions written between FOR and NEXT a specific number of times. The number of times the loop is repeated is determined by the values of <expression 1>, <expression 2>, and <expression 3>, with <variable> used as a counter to keep track of the number of loops made.

The initial value of <variable> is determined by the value of <expression 1>, and <expression 2> indicates the value of <variable> on which the loop is to be terminated.
<expression 3> indicates the value by which <variable> is to be incremented each time the loop is repeated. MFBASIC checks the value of <variable> before executing the instructions in the loop, then the instructions in the loop are executed if <variable> is less than or equal to <expression 2>; otherwise, the loop is terminated and execution proceeds with the statement following NEXT.

An increment of "1" is assumed if STEP is not specified; however, a negative value must be specified as the STEP in <expression 3> if the starting value (<expression 1>) is greater than the ending value (<expression 2>) (otherwise, the instructions following FOR will not be executed).

FOR...NEXT loops may be nested; that is, one FOR...NEXT loop may be included within the range of another. When loops are nested, a different variable name must be specified for <variable> of each loop. Further, the NEXT statement for the inner loop must appear before that for the outer one.

If nested loops end at the same point, a single NEXT statement may be used for all of them. In this case, the variable names must be specified following NEXT in the reverse order in which they appear in the FOR statements of the nested loops; in other words, the first variable name following next must be that which is specified in the

ClibPDF - www.fastio.com

nearest preceding FOR statement, the second variable name following NEXT must be that which is specified in the next nearest preceding FOR statement, and so forth.

If a NEXT statement is encountered before its corresponding FOR statement, a "NEXT without FOR" message is displayed and execution is terminated. If a FOR statement without a corresponding NEXT statement is encountered, a "FOR without NEXT" message is displayed and execution is terminated.

See also WHILE...WEND

Example 1

```
10 PRINT"X","X^2","X^3"
20 FOR X=0 TO 10
30 PRINT X,X^2,X^3
40 NEXT
```

```
RUN
X             X^2           X^3
 0             0             0
 1             1             1
 2             4             8
 3             9             27
 4             16            64
 5             25            125
 6             36            216
 7             49            343
 8             64            512
 9             81            729
 10            100           1000
Ok
```


ClibPDF - www.fastio.com

Example 2

```
10 FOR A=1 TO 2
20 FOR B=1 TO 2
25 FOR C=1 TO 2
30 PRINT A,B,C
40 NEXT C,B,A
```

```
RUN
 1             1             1
 1             1             2
 1             2             1
 1             2             2
 2             1             1
 2             1             2
 2             2             1
 2             2             2
Ok
```

Example 3

```
10 I=3
20 FOR I=1 TO I+7
30 PRINT I;
40 NEXT
50 END
```

```
RUN
 1  2  3  4  5  6  7  8  9  10
Ok
```

In Example 3 above, the loop is repeated ten times because the loop variable's final value is always set before the initial value.

ClibPDF - www.fastio.com

GCURSOR

Format GCURSOR [STEP](horizontal position, vertical position), (<variable 1>, <variable 2>)

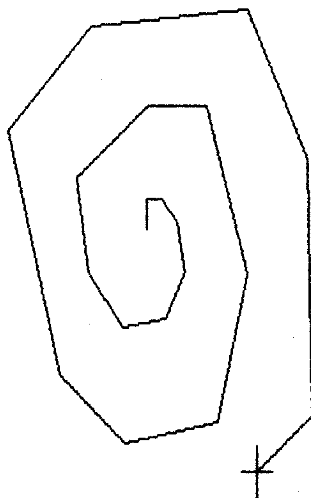
Purpose The GCURSOR statement displays the graphic cursor and reads its coordinates into variables.

Remarks <horizontal position> and <vertical position> are graphic coordinates which specify the initial position in which the graphic cursor is to be displayed. These coordinates can be specified either as absolute values or, by adding the STEP function, as relative values. (See the explanation of the CIRCLE statement for an example of use of the STEP function.)

The graphic cursor is displayed as a large " + " mark, and can be moved by pressing the cursor control keys. The horizontal and vertical positions of the graphic cursor are read into <variable 1> and <variable 2>, respectively, when the RETURN key is pressed. The horizontal position must be specified as an integer expression whose value is between 0 and 639, and the vertical position must be specified as an integer expression whose value is between 0 and 399.

Example

```
10 CLS
20 GCURSOR (32,200),(X1,Y1)
30 GCURSOR (X1,Y1),(X2,Y2)
40 LINE (X1,Y1)-(X2,Y2)
50 X1=X2
60 Y1=Y2
70 GOTO 30
```



NOTE:
After execution of the GCURSOR statement, the last reference pointer used by STEP is updated to (horizontal position, vertical position).

ClibPDF - www.fastio.com

GET

Format GET[#] <file number>[,<record number>]

Purpose The GET statement reads a record into a random file buffer from a random disk file.

Remarks This statement reads a record into the random file buffer from the file OPENed under the integer expression specified for <file number>. <record number> must also be specified as an integer expression; if omitted, the record read is that following the record read by the previous GET statement. The highest record number which can be read is 32767.

See also FIELD, LSET/RSET, OPEN, PUT

Example See Chapter 5.

ClibPDF - www.fastio.com

GET@

Format

GET[@] (horizontal position 1,vertical position 1)
-[STEP](horizontal position 2,vertical position 2),<array name>

Purpose

This statement reads the settings of the specified range of display dots into a variable array.

Remarks

The range of dot settings read into the variable array is that of the rectangular area defined by a diagonal line between (horizontal position 1,vertical position 1) and (horizontal position 2,vertical position 2).

(horizontal position 1,vertical position 1) must be specified as absolute coordinates; however, relative coordinates can be specified for (horizontal position 2,vertical position 2) by specifying STEP. (See CIRCLE for an example of coordinate specification with STEP.) The values of numeric expressions specifying the horizontal positions must be between 0 and 639, and those for the vertical positions must be between 0 and 399.

The array in which the dot settings are to be stored must be defined in advance by executing a DIM statement. The maximum values which must be specified for the array subscripts in the DIM statement are calculated as shown below.

First, calculate the number of bytes (A) required for storage of the dot settings by GET@ (X1,Y1)-(X2,Y2) <array variable> as follows.

A=4+((<no. of horizontal dots>+7)\8)*<no. of vertical dots>*M

In the above, M=1 for the black and white mode, and M=3 for the color mode. The maximum subscript value which must be specified in the DIM statement is equal to A\N+1, where N is 2 for an integer type array, 4 for a single precision array, and 8 for a double precision array.

After execution of the GET@ statement, the last reference pointer used by STEP is updated to (horizontal position 2, vertical position 2).

See also

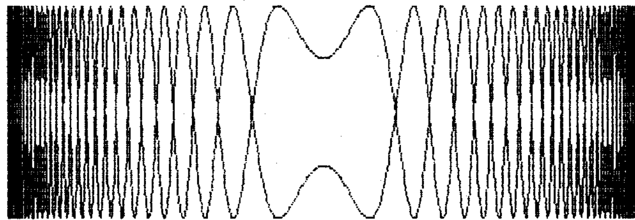
PUT@

ClibPDF - www.fastio.com

Example

```
10 CLS
20 HL=300:HH=300:VL=200:VH=200
30 HL=SIN(-3.14159)*250+300:VL=SIN(COS(-3.14159)*1
00)*50+200:HH=HL:VH=VL
35 PSET (HL,VL)
40 FOR X=-3.14159 TO 3.14159 STEP 3.14159/900
50 H1=INT(SIN(X)*250+300):V1=INT(SIN(COS(X)*100)*5
0+200)
60 IF HL>H1 THEN HL=H1
70 IF HH<H1 THEN HH=H1
80 IF VL>V1 THEN VL=V1
90 IF VH<V1 THEN VH=V1
100 LINE -(H1,V1)
110 NEXT
120 A=4+((HH-HL+8)\8)*(VH-VL+1)*3
130 DIM B#(A\8+1)
140 GET@(HL,VL)-(HH,VH),B#
150 CLS
160 PUT@(30,30),B#
170 PUT@(60,150),B#,PRESET

Ok
```



ClibPDF - www.fastio.com

GOSUB...RETURN

Format

GOSUB <line number>

.

.

.

RETURN

Purpose

The GOSUB and RETURN statements are used to branch to and return from subroutines.

Remarks

The GOSUB statement transfers execution to the program line number specified in <line number>; execution is returned to the point following that at which the subroutine call was made when a RETURN statement is encountered. Subroutines may include more than one RETURN statement if the program logic dictates a return from different points in the subroutine.

A subroutine may be called any number of times in a program, and one subroutine may be called by another. Nesting of subroutines in this manner is limited only by the amount of stack space available for storing return addresses. An "Out of memory" error will occur if the stack space is exceeded, as may be demonstrated by executing the example below.

Example 1

```
10 X=0
20 X=X+1
30 PRINT X;
40 GOSUB 20
Ok
RUN
 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22
 23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42
 43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62
 63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82
 83  84  85
Out of memory in 20
Ok
```


ClibPDF - www.fastio.com

Subroutines may be located anywhere in a program; however, it is recommended that they be made readily distinguishable from the main program. Care must be taken to ensure that execution does not move into a subroutine without its being called. This can be avoided with the STOP, END, or GOTO statements; the STOP and END statements stop execution when encountered, while the GOTO statement makes it possible to route execution around a subroutine. If a RETURN statement is encountered without a corresponding GOSUB statement, a "RETURN without GOSUB" error will occur.

Example 2

```
10 INPUT"INPUT ANGLE IN DEGREES";N
20 GOSUB 40
30 GOTO 10
40 P=COS(N*3.14159/180)
50 Q=SIN(N*3.14159/180)
60 PRINT "N=";N,"COS(N)=";P,"SIN(N)=";Q
70 RETURN
```

```
RUN
INPUT ANGLE IN DEGREES? 0
N= 0          COS(N)= 1     SIN(N)= 0
INPUT ANGLE IN DEGREES? 15
N= 15         COS(N)= .965926         SIN(N)= .258819
INPUT ANGLE IN DEGREES? 30
N= 30         COS(N)= .866026         SIN(N)= .5
INPUT ANGLE IN DEGREES? 45
N= 45         COS(N)= .707107         SIN(N)= .707106
INPUT ANGLE IN DEGREES?
```


ClibPDF - www.fastio.com

GOTO

Format GOTO <line number>

Purpose The GOTO statement unconditionally transfers program execution to the program line specified by <line number>.

Remarks This statement is used to make unconditional "jumps" from one point in a program to another. If the statement(s) on the line specified by <line number> is an executable statement (other than a REM or DATA statement), execution resumes with that statement; otherwise, execution resumes with the first executable statement encountered following <line number>. An "Undefined line number" error will occur if <line number> refers to a non-existent line.

Example

```
10 READ A,B
20 IF A=0 AND B=0 THEN 80
30 PRINT "A=";A,"B=";B
40 S=A*B
50 PRINT "PRODUCT IS";S
60 GOTO 10
70 DATA 12,5,8,3,9,0,0,0
80 END
```

```
RUN
A= 12         B= 5
PRODUCT IS 60
A= 8          B= 3
PRODUCT IS 24
A= 9          B= 0
PRODUCT IS 0
Ok
```


ClibPDF - www.fastio.com

IF...THEN [...ELSE]/IF...GOTO

Format

IF <logical expression>
| THEN | <statement> / <line No.> | [ELSE | <statement> / <line No.>] |
| GOTO | <line No.> | | |

Purpose

This statement is used to change the flow of program execution according to the results of a logical expression.

Remarks

The THEN or GOTO clause following <logical expression> is executed if the result of <logical expression> is non-zero (true). Otherwise, the THEN or GOTO clause is ignored and the ELSE clause (if any) is executed; execution then proceeds with the next executable statement. With the first format, the THEN clause may be followed by a line number or one or more statements. Specifying a line number after THEN causes program execution to be transferred to that program line in the same manner as with GOTO. With the second format, a line number is always specified following GOTO.
With MFBASIC, IF...THEN...ELSE statements may be nested by including one such statement as a clause in another. Such nesting is limited only by the maximum length of the program line.

For example, the following is a correctly nested IF...THEN statement:

```
20 IF X>Y THEN PRINT "X IS LARGER THAN Y" ELSE IF Y>X
THEN PRINT "X IS SMALLER THEN Y" ELSE PRINT " X
EQUALS Y"
```

If a statement contains more THEN than ELSE clauses, each ELSE clause is matched with the nearest preceding THEN clause. For example, the following statement will display "A=C" when A=B and B=C; "A< >C" if A=B and B< >C; and nothing at all if A< >B.

```
IF A=B THEN IF B=C THEN PRINT "A=C"
ELSE PRINT "A< >C"
```

NOTE:
When using IF together with a relational expression which tests equality, remember that the internal representation of values which result from floating point computations are not always exact. Therefore, it is preferable to test computed values against the range over which the accuracy of such values may vary. For example, when testing for equality between "1" and a computed variable A, the following form is recommended.

```
10 IF ABS(A-1.0)<1.0E-6 THEN...
```


ClibPDF - www.fastio.com

The statement(s) following THEN will be executed if the value of A is within ±1.0E-6 of 1.0.

Example 1

```
100 IF (A>1) and (A<10) THEN X=SQR(Y):GOTO 150
110 X=Y
```

This example calculates X = SQR(Y) and branches to program line 150 if the value of A is greater than 1 and less than 10; otherwise, execution continues with line 110.

Example 2

```
100 IF X=Y THEN PRINT A$ ELSE LPRINT A$
```

This statement outputs the value of variable A$ to either the display or printer, depending on whether the value of X equals Y. If X equals Y, output is to the display; otherwise, output is to the printer.

ClibPDF - www.fastio.com

INPUT

Format

INPUT[;]<list of variables>
INPUT[;]<"prompt string"><;|,><list of variables>

Purpose

This statement makes it possible to substitute values into variables from the keyboard during program execution.

Remarks

Program execution pauses when an INPUT statement is encountered to allow data to be substituted into variables from the keyboard. One data item must be typed in for each variable name specified in <list of variables>. After all data items have been typed in, they are substituted into the variables by pressing the RETURN key.

With the first format, a question mark is displayed to indicate that the program is waiting for data entry; with the second, the specified prompt string is displayed. The prompt string will be followed by a question mark if it is followed by a semicolon; if it is followed by a comma, the question mark will be suppressed. With either format, a semicolon following INPUT causes the carriage return/line feed sequence to be suppressed when the RETURN key is pressed to enter data; this prevents the cursor from advancing to the next line.

When more than one variable name is specified in <list of variables>, each variable name must be separated from the following one by a comma. When such an INPUT statement is executed, commas are also used to separate the data items typed in from the keyboard. The items entered in response to an INPUT statement are substituted into the variables specified in <list of variables > when the RETURN key is pressed.

The variables specified in <list of variables> may be either numeric or string variables (including array varibles); however, the type of each data item entered must match that of the corresponding variable. A string variable must be included in quotation marks if any commas (or leading or trailing spaces) are to be included in the string; otherwise, quotation marks are not required.

If the response to an INPUT statement includes other than the correct number or type of data items, the message "?Redo from start" will be displayed, followed by the prompt string (if any). When this occurs, reenter the data items correctly, then press the RETURN key; no values will be substituted into variables until an acceptable response is made.

ClibPDF - www.fastio.com

Example 1

```
10 ON ERROR GOTO 60
20 INPUT "INPUT ARBITRARY NUMBER";X
30 Y=SQR(X)
40 PRINT "SQUARE ROOT OF";X;"IS";Y
50 GOTO 20
60 IF ERL=30 AND X<0 THEN PRINT "Negative number
error - Redo":RESUME 50
```

```
RUN
INPUT ARBITRARY NUMBER? 1
SQUARE ROOT OF 1 IS 1
INPUT ARBITRARY NUMBER? 5
SQUARE ROOT OF 5 IS 2.23607
INPUT ARBITRARY NUMBER? 7
SQUARE ROOT OF 7 IS 2.64575
INPUT ARBITRARY NUMBER? 9
SQUARE ROOT OF 9 IS 3
INPUT ARBITRARY NUMBER? -8
Negative number error - Redo
INPUT ARBITRARY NUMBER?
```

Example 2

```
10 INPUT "ENTER ANY TWO STRINGS";A$,B$
20 PRINT A$;" ";B$
```

```
RUN
ENTER ANY TWO STRINGS? EPSON,QX-10
EPSON QX-10
Ok
```


ClibPDF - www.fastio.com

INPUT

Format INPUT # <file number>, <variable list>

Purpose This statement is used to read data items into variables from a sequential file on a flexible disk.

Remarks The sequential file from which data items are to be read must have been previously opened for input by executing an OPEN statement. The <file number> specified following INPUT # is that under which the file was OPENed, and <variable list> specifies the names of variables into which data items are to be read. Data items read must be of the same type as the variables into which they are substituted.

When a sequential file is read with the INPUT # statement, the first character encountered which is not a space is assumed to be the start of a data item. With string items, the end of each item is assumed when the next subsequent comma or carriage return is encountered (unless the item is enclosed in quotation marks, in which case the item is assumed to include all enclosed characters). String data is also automatically delimited wherever 255 characters have been input.

With numeric items, the end of each item is assumed when a space, comma, or carriage return is encountered. Therefore, care must be taken to ensure that proper delimiters are used when the file is written to the disk with the PRINT # statement; see the explanation of the PRINT # statement and Chapter 5 for further information.)

See also INPUT, LINE INPUT #, OPEN, PRINT #, WRITE, WRITE #

Example 1 See Chapter 5.

ClibPDF - www.fastio.com

KEYn/KEY LIST/KEY LLIST

Format

```
KEY<n>, <X$>
KEY LIST
KEY LLIST
```

Purpose

The KEY statement is used to define or list the functions of the user programmable function keys.

Remarks

The first format is used to define the functions of function keys. Here, <n> is an integer from 1 to 10 which indicates the number of the function key being defined, and <X$> is a character string of up to 15 characters which is to be assigned to that key as a function. For example,

```
KEY 1, "LIST"
```

assigns the LIST function to programmable function key 1.

A control code can be included in the character string assigned to the function key by adding +CHR$(I) to X$, where I is the ASCII code for that control character. For example,

```
KEY 1, "LIST"+CHR$(13)
```

will assign the character string "LIST" plus a return code to programmable function key 1; subsequently, pressing F1 will list the entire contents of any program currently stored in memory.

KEY LIST and KEY LLIST output a list of the current function key definitions to the display and printer.

Example

```
KEY 1, "SAVE"
```


ClibPDF - www.fastio.com

KILL

Format KILL <file descriptor>

Purpose The KILL command is used to delete files from the disk in one of the QX-10's flexible disk drives.

Remarks The KILL command can be used to delete any type of disk file. The full file descriptor must be specified if the file to be deleted is on a disk in a drive other than that which is currently active. Otherwise, only the primary file name and extension need to be specified.

Example 1 KILL "FILE3.BAS"

Example 2 KILL "B:SAMPLE1.BAS"

NOTE:
Operation of the KILL command is not assured if it is issued against a file which is currently OPEN.

ClibPDF - www.fastio.com

LET

Format [LET] <variable> = <expression>

Purpose This statement is used to assign the value of an expression into a variable.

Remarks Note that the word LET is optional. Thus, the two examples below give the same result.

Example 1

```
10 LET A=3
20 LET B=5
30 LET C=A*B
40 PRINT C
50 END

RUN
 15
Ok
```

Example 2

```
10 A=3
20 B=5
30 C=A*B
40 PRINT C
50 END

RUN
 15
Ok
```


ClibPDF - www.fastio.com

LINE

Format LINE [[STEP](X1,Y1)]-[STEP](X2,Y2)[,[<color code>][,[B[F]][,<line style>]]]

Purpose This statement draws a straight line between two specified points.

Remarks The straight line is drawn between the points whose graphic coordinates are specified by (X1,Y1) and (X2,Y2). Relative coordinates can be specified for either or both the starting and ending points by adding STEP. If (X1,Y1) is omitted, the last coordinates used by the preceding PSET, PRESET, LINE, CONNECT, GET@, or PUT@ statement are assumed; this is the same as specifying STEP(0,0).

<color code> specifies the color to be used in drawing the line; if omitted, the current foreground color is assumed as the default value.

When the B option is specified, this statement draws a rectangle whose diagonal dimension is defined by the two points specified. If the F option is specified together with the B option, the rectangle is painted; however, the BF option cannot be specified if a <line style> specification is made.

The <line style> option specifies the style of line to be drawn; any 16-bit number from &H0000 to &HFFFF can be specified in this option. The dot settings in the line drawn will have a one-to-one correspondence with the settings of the 16 bits of the value specified in the <line style> option; i.e., dots corresponding to "1" bits will be set, while those corresponding to "0" bits will not be set. If the line drawn is longer than 16 dots, the dot sequence will be repeated in each 16-dot section.

See also CIRCLE, COLOR, CONNECT, PRESET, PRESET

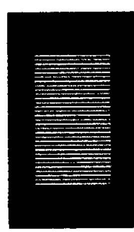
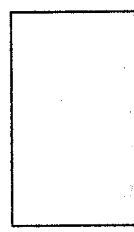
NOTE:
After execution of the LINE statement, the last reference pointer is updated to the values specified for (X2, Y2)

ClibPDF - www.fastio.com

Example 1

```
10 CLS
20 LINE (100,100)-(200,200),,BF
30 LINE (250,100)-(350,200),,BF
40 FOR I=120 TO 180 STEP 2
50 LINE (270,I)-(330,I),0
60 NEXT I
70 LINE (400,100)-(500,200),,B
80 END

Ok
```

ClibPDF - www.fastio.com

Example 2

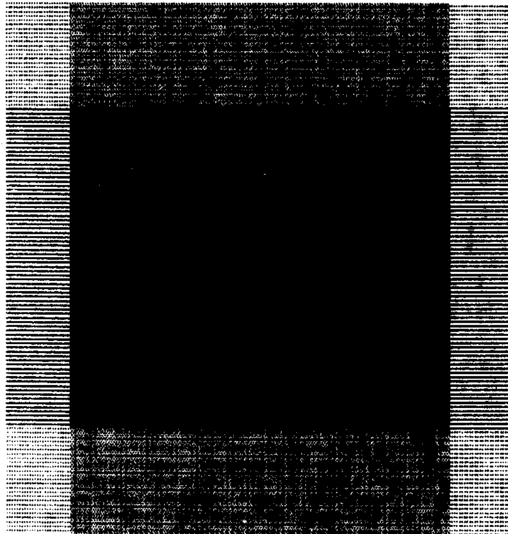
```
10 CLS
20 PRINT"LINE STYLE","SAMPLE"
25 Y=30
30 FOR I=1 TO 153 STEP 9
40 PRINT "&H";HEX$(I)
50 LINE (100,Y)-(300,Y),,,I
60 Y=Y+20
70 NEXT I
```

```
LINE STYLE   SAMPLE
&H1          . . . . . . . . . . . . . .
&HA          .. . . . . . . . . . . .
&H13         .. .. .. .. .. .. .. .. .. .. ..
&H1C         . . . . . . . . . . . .
&H25         -- -- -- -- -- -- -- -- -- --
&H2E         - - - - - - - - - - - -
&H37         - - - - - - - - - - - - .
&H40         . . . . . . . . . . . .
&H49         ... ... ... ... ... ... ... ...
&H52         .. .. .. .. .. .. .. .. .. ..
&H5B         - - - - - - - - - - - - .
&H64         .. .. .. .. .. .. .. .. .. ..
&H6D         - - - - - - - - - - - - .
&H76         - - - - - - - - - - - -
&H7F         - - - - - - - - - - - - .
&H88         .. .. .. .. .. .. .. .. .. ..
&H91         ... ... ... ... ... ... ... ... .
Ok
```


ClibPDF - www.fastio.com

Example 3

```
10 CLS
20 LOCATE 1,20,1
30 INPUT"enter color numbers a,b,c,d";A,B,C,D
40 FOR X=1 TO 350 STEP 2
50 LINE (X,1)-(X,200),A,,&H5555
60 LINE (X+51,1)-(X+51,200),B,,&HAAAA
70 NEXT X
80 FOR X=2 TO 350 STEP 2
90 LINE (X,50)-(X,250),C,,&HAAAA
100 LINE (X+51,50)-(X+51,250),D,,&H5555
110 NEXT X
120 INPUT A:GOTO 10
```



```
enter color numbers a,b,c,d? 2,3,4,5
?
```


ClibPDF - www.fastio.com

LINE INPUT

Format LINE INPUT[;][<"prompt string">;]<string variable>

Purpose The LINE INPUT statement is used to substitute character strings into string variables from the keyboard during program execution.

Remarks As with the INPUT statement, the LINE INPUT statement is used to substitute values into variables from the keyboard. However, this statement is only applicable to character strings and string variables, and only one such string can be input at a time (it is not possible to specify a list of variables); the maximum length of the input line is 255 characters.

Another difference between the INPUT and LINE INPUT statements is that, whereas the former requires that the string entered be enclosed in quotation marks if it includes any commas, the latter substitutes all characters entered into the specified variable. Further, no question mark is displayed when a LINE INPUT statement is executed unless one has been included in <"prompt string"> by the user.

As with the INPUT statement, a semicolon following LINE INPUT suppresses the carriage return/line feed code so that the cursor remains on the same line.

See also INPUT

Example

```
10 LINE INPUT "CHARACTER INPUT ";A$
20 PRINT A$
30 END
```

```
RUN
CHARACTER INPUT ABC ,DE;"FGH :
ABC ,DE;"FGH :
Ok
```


ClibPDF - www.fastio.com

LINE INPUT

Format LINE INPUT # <file number>,<string variable>

Purpose As with the INPUT # statement, the LINE INPUT # statement is used to read data items from a disk file into variables.

Remarks The LINE INPUT # statement differs from the INPUT # statement in that it inputs all characters in the sequential file up to the first carriage return encountered before substituting them into <string variable> (all other delimiting characters recognized by the INPUT # statement are regarded as part of the string being read); the carriage return code itself is skipped, so that the next LINE INPUT # statement reads characters starting with the first character following the carriage return. This statement can be used to read all values written by a PRINT # statement into one variable; it also allows lines of an MFBASIC program which has been saved in ASCII format to be input as data by another program.

Example See Chapter 5.

ClibPDF - www.fastio.com

LIST

Format

LIST [<line number>[-[<line number>]]]
LIST <file descriptor>[,<line number>[-[<line number>]]]

Purpose

This command is used to list of all or part of an MFBASIC program on the display screen.

Remarks

Executing the LIST command without specifying line numbers or a file descriptor causes the lines of the program currently contained in memory to be output to the CRT screen.

If an asterisk is appended to LIST, the LIST is output without line numbers.

For other formats, the program lines listed and the device to which the list is output are as follows.

LIST <line number>-<line number>
Lists the program currently contained in memory to the CRT screen, starting with the first <line number> and ending with the second.

LIST <line number>-
Lists the program currently in memory to the CRT screen, starting with the specified line and ending with the last line of the program.

LIST -<line number>
Lists all lines of the program from the first line to that specified in <line number>.

LIST <line number>
Lists the program line specified in <line number>.

LIST <file descriptor>
Outputs the program in memory to the device specified in <file descriptor> in ASCII format. (This is the same as using the SAVE statement with the A option to output a program to the specified device in ASCII format.) Devices which can be specified include the CRT, RS-232C interface, flexible disk drives, and line printer. If line numbers are specified, only the specified lines are output.

When a program is being listed, the listing can be terminated before execution of the command is completed by pressing the BREAK or CRTL and C keys. MFBASIC always returns to the command level after execution of a LIST command.

ClibPDF - www.fastio.com

Example 1 LIST
Lists the program currently in memory.

Example 2 LIST*
Same as above, but outputs the program list without line numbers.

Example 3 LIST 500
Lists line 500 of the program currently in memory.

Example 4 LIST 150-
Lists all lines from line 150 to the end of the program in memory.

Example 5 LIST -1000
Lists all lines from the beginning of the program in memory through line 1000.

Example 6 LIST 150-1000
Lists program lines from 150 through 1000.

Example 7 LIST "CMOS:"
SAVEs the program in memory to the CMOS RAM file in ASCII format.

ClibPDF - www.fastio.com

LLIST

Format LLIST [<line number>[-[<line number>]]]

Purpose This command lists all or part of the program in memory to the printer.

Remarks The LLIST command is used in the same manner as LIST, but output is always directed to the printer connected to the QX-10. MFBASIC always returns to the command level after execution of a LLIST command.

Example See LIST.

ClibPDF - www.fastio.com

LOAD

Format LOAD <file descriptor>[,R]

Purpose This command loads a program file into memory from a disk drive, disk image RAM, the RS-232C interface, or "CMOS:" file.

Remarks Specify the device name, primary file name, and extension under which the file was SAVEd in <file descriptor>. If the device name is omitted, the currently active drive is assumed; if the extension file name is omitted, ".BAS" is assumed.

When a LOAD command is executed, all files currently open are closed and all variables and program lines currently resident in memory are cleared before the specified program is loaded. However, if the "R" option is appended to the LOAD command, all data files are left open and the program is RUN as soon as loading is completed; therefore, LOAD with the "R" option may be used to chain programs (or segments of the same program).

When programs are chained in this manner, information may be passed betweem them by storing it in disk data files.

See also CHAIN, MERGE, RUN, SAVE

Example 1 LOAD "LNINPT"

Example 2 LOAD "B:LNINPT.BAS"

ClibPDF - www.fastio.com

LOCATE

Format LOCATE [<X>][,[<Y>][,<cursor switch>]]

Purpose This statement moves the cursor to a specific position on the display screen.

Remarks The LOCATE statement moves the cursor to the character screen coordinates specified by <X>, <Y>.In the WIDTH 80 mode, the value specified for <X> must be in the range from 1 to 80, with 1 indicating the screen column on the far left; in the WIDTH 40 mode, the value specified for <horizontal position> must be in the range from 1 to 40. In either mode, the value specified for <Y> must be in the range from 1 to 20, with 1 indicating the top line of the screen.

Specifying 0 in the <cursor switch> option turns off the cursor, and specifying 1 turns the cursor on.

The cursor switch is forcibly set to 0 when MFBASIC returns to the command level; further, regardless of the setting of the cursor switch, the cursor is displayed when INPUT or LINE INPUT statements or the INPUT$ function are encountered during program execution.

Example

```
10 CLS
20 A$="ABCDEFG"
30 LOCATE 5,2
40 PRINT A$
50 LOCATE 10,3
60 PRINT A$
70 LOCATE 15,4
80 PRINT A$
90 END
```

```
    ABCDEFG
         ABCDEFG
              ABCDEFG
Ok
```


ClibPDF - www.fastio.com

LPRINT/LPRINT USING

Format LPRINT [<list of expressions>]
LPRINT USING <"format string">; <list of expressions>

Purpose These statements are used to print data on the printer connected to the QX-10.

Remarks These statements are used in the same manner as the PRINT and PRINT USING statements, but output is directed to the printer instead of the display screen.

See also PRINT, PRINT USING

ClibPDF - www.fastio.com

LSET/RSET

Format

LSET <string variable> = <string expression>
RSET <string variable> = <string expression>

Purpose

These statements move data into a random file buffer to prepare it for storage in a random access file with the PUT statement.

Remarks

If the length of <string expression> is less than the number of bytes in the random file buffer which were FIELDed to <string variable>, the LSET statement left-justifies the string data in the field and the RSET statement right-justifies it. Extra spaces in the random file buffer are padded with blanks. If the length of <string expression> is greater than the number of bytes FIELDed to <string variable>, characters are dropped from the right end of <string expression> when it is moved into the buffer.

Numeric values must be converted to strings before they are LSET or RSET. See the explanations of the MKI$, MKS$, and MKD$ functions in Chapter 4 for conversion of numeric values to strings.

See also

FIELD, GET, OPEN, PUT

Example

```
10 A$=STRING$(20," ")
20 N$="EPSON"
30 RSET A$=N$
40 PRINT N$
50 PRINT A$
Ok
RUN
EPSON
               EPSON
Ok
```

NOTE:
The LSET and RSET statements can also be used to left or right justify a string in a non-fielded string variable. For example, the following program lines right justify string N$ in a 20-character field prepared in variable A$. This can be very useful for formatting printed output.

ClibPDF - www.fastio.com

MERGE

Format MERGE <file descriptor>

Purpose This command merges a program from a disk drive, disk image RAM, RS-232C interface, or the "CMOS:" file with the program currently in memory.

Remarks Specify the device name, primary file name, and extension under which the file was SAVEd in <file descriptor>. If the device name is omitted, the currently active drive is assumed; if the extension is omitted, ".BAS" is assumed. The file being merged must have been SAVEd in ASCII format. (Otherwise, a "Bad file mode" error will occur.)

If any lines of the program being merged have the same numbers as lines of the program in memory, the merged lines will replace the corresponding lines in memory. Thus, MERGEing may be thought of as "inserting" program lines from a storage device into the program in memory.

MFBASIC always returns to the command level after execution of a MERGE command.

See also SAVE

Examples

```
MERGE "TEST1"
MERGE "CMOS:TEST1.BAS"
```


ClibPDF - www.fastio.com