Chapter 3 MFBASIC COMMANDS AND STATEMENTS

This chapter describes the commands and statements used with MFBASIC, as well as those system variables that can be used as statements.

Commands and statements are the instructions which are used to control operation of the computer, and are the basic unit of which programs are constructed. The distinction between commands and statements is that the former are generally entered in the direct mode to control operation of programs, while statements are instructions which are entered in the indirect mode as the units making up programs themselves. In fact, however, most of the commands and statements described in this chapter can be entered in either mode.

System variables are variables whose names are included in the list of reserved words and whose contents are updated automatically by the operating system. Since the system variables in themselves do not affect operation of the computer, they cannot be regarded as statements in the strict sense of the word. However, the values of some system variables can be changed arbitrarily by specifying them in assignment statements, and thus descriptions of these variables are included in this Chapter.

The format used in describing each of the commands, statements, and variables included in this Chapter is as follows.

Format The format section of each description illustrates the general format for specification of that command or statement. The meanings of the symbols used in the general format descriptions are as described in "Format Notations" below.

Purpose Explains the purpose of the command or statement.

Remarks Gives detailed instructions for using the command/statement.

See also Refers the user to the descriptions of other commands/statements whose operation is related in some way to that of the command/statement being described.

Example Presents a sample program or program segment showing use of the command/statement.

NOTE:
Outlines precautions which should be observed in using the command/statement or presents other related information.

Format Notations

The following rules apply to specification of commands or statements.

(1) Items shown in capital letters are MFBASIC reserved words used in the commands/statements, and must be input letter for letter exactly as shown. Any combination of upper- or lowercase letters can be used to enter reserved words (MFBASIC automatically converts lowercase letters to uppercase except when they are included between quotation marks or as part of a remark statement).

(2) Angle brackets (< >) indicate items which must be specified by the user. Specification of such items is mandatory unless they are enclosed in square brackets ([ ]). (The brackets themselves have no meaning in the command/statement, and should not be specified).

(3) Square brackets ([ ]) indicate items whose specification is optional.

(4) All punctuation marks (commas, parentheses, semicolons, hyphens, equal signs, and so forth) must be included in the command/statement exactly as shown.

(5) Items followed by an ellipsis (...) may be repeated any number of times (up to the maximum length of the logical line).

(6) A vertical bar indicates mutually exclusive items; specify only one of the items shown.

AUTO

Format AUTO [<line number>[,[<increment>]]]

Purpose This command is entered in the direct mode to initiate automatic generation of program line numbers during program entry.

Remarks Numbering begins at <line number> and is incremented by <increment> for each subsequent program line. The default for both values is 10. If <line number> is followed by a comma but <increment> is not specified, the increment specified by the preceding AUTO command is assumed.

If a program line has already been entered for a line number generated by the AUTO numbering function, an asterisk (*) is displayed immediately after the number to warn the user that the line contains statements. Pressing the RETURN key at this time causes that line to be skipped without affecting the current contents of the line; entering any characters before pressing the RETURN key causes the former contents of the line to be replaced with the characters entered.

The AUTO numbering function can be terminated and MFBASIC returned to the command level by pressing CTRL and C or the BREAK key.

Example 1 AUTO

Generates line numbers in increments of 10 starting with line number 10. (10, 20, 30,...)

Example 2 AUTO 100,50

Generates line numbers in increments of 50 starting with line number 100. (100, 150, 200,...)

BEEP

Format

BEEP [<duration>]
ON
OFF

Purpose

This statement controls operation of the QX-10's sound generator.

Remarks

This statement sounds the speaker built into the keyboard of the QX-10. BEEP ON sounds a continuous tone of infinite duration, and BEEP OFF stops the tone. <duration> is specified as a value from 0 to 255, where values from 1 to 254 sound a tone whose duration is equal to <duration> × 10ms. Specifying 0 for <duration> has the same effect as executing BEEP OFF, and specifying 255 has the same effect as executing BEEP ON. BEEP 10 is assumed if this statement is executed without specifying <duration>, ON, or OFF.

See also

SOUND

Example 1

BEEP ON Generates a continuous tone.

Example 2

BEEP OFF Turns off the sound generator.

Example 3

BEEP 100 Generates a 1-second tone.

NOTE:
The tone produced by executing the BEEP statement cannot be terminated by pressing the BREAK key; stop the sound generator by executing BEEP OFF.

BIT

Format

BIT [ON | OFF]

Purpose

This statement makes it possible to select the bit image mode for output to the printer.

Remarks

The BIT statement changes the mode of output to the printer. When MFBASIC is started, it is in the BIT OFF mode. In this mode, graphic characters (characters with internal codes from &H7F to &H9F) and 2-byte characters can be output to the printer. However, in this mode control codes other than bell (&H07), line feed (&H0A), form feed (&H0C), carriage return (&H0D), and escape (&H1B) are ineffective. Further, all escape sequences other than those built into CP/M's basic input/output system are ineffective, and cannot be output to the printer.

In the BIT ON mode, statements such as PRINT and PRINT # output all codes to the printer as is. Therefore, graphic codes output to the printer are printed using the character generator built into the printer, and 2-byte characters are printed as two 1-byte characters. However, this mode makes it possible to use all of the control codes and escape sequences, and therefore the BIT ON mode must be used for bit image output.

NOTE:
It is not possible to return to the BIT OFF mode by pressing the BREAK key; the BIT ON mode must be terminated by entering a BIT OFF command.

Example

```
10 BIT ON
20 '
30 'LINE 70 BELOW PLACES THE EPSON PRINTER IN THE HIGH
40 'RESOLUTION GRAPHICS MODE AND RESERVES 90 COLUMNS
50 'FOR GRAPHICS
60 '
70 LPRINT CHR$(27);CHR$(75);CHR$(90);CHR$(0);
80 '
90 'THE FOR-NEXT LOOP BELOW FIRES THOSE PINS IN THE
100 'PRINT HEAD WHICH (FROM BOTTOM TO TOP) CORRESPOND
110 'TO THE "1" BITS OF THE BINARY EQUIVALENT OF 85.
120 '
130 FOR I=0 TO 89
140 LPRINT CHR$(85);
150 NEXT I
160 BIT OFF
170 LPRINT CHR$(13)
180 LPRINT "Epson BASIC"
190 BIT ON
200 LPRINT CHR$(27);CHR$(75);CHR$(90);CHR$(0);
210 FOR I=0 TO 89
220 LPRINT CHR$(85);
230 NEXT I
240 BIT OFF
250 LPRINT

Ok
RUN


Epson BASIC
```

CALL

Format CALL <variable name>[(<argument list>)]

Purpose This statement is used to call machine language programs.

Remarks The CALL statement is one method of transferring program execution to a machine language subroutine. (See also the discussion of the USR function in Chapter 4.) <variable name> is the name of the variable which indicates the the machine language program's starting address in memory (this may not be an array name). <argument list> is the list of parameters which is passed to the machine language subroutine by the calling program. See Appendix E for further details on use of the CALL statement.

See also USR, Appendix E

Example

```
10 CLEAR ,&HBFFF
20 ADRS=&HC000
30 FOR I=1 TO 10
40 READ A
50 POKE ADRS,A
60 ADRS=ADRS+1
70 NEXT I
80 DATA &H3A,&H09,&HC0,&HC6,&H01,&H32,&H09,&HC0,&H
C9,&H00
90 ADRS=&HC000
100 INPUT"INPUT NUMBER FROM 1 TO 254";B
110 POKE &HC009,B
120 CALL ADRS
130 C=PEEK(&HC009)
140 LPRINT B;"+ 1=";C
150 GOTO 100

 50 + 1= 51
 100 + 1= 101
 150 + 1= 151
 200 + 1= 201
```

CHAIN

Format CHAIN [MERGE] <filename>[,[<line number exp>] [,ALL] [,DELETE<range>]]

Purpose This statement calls a BASIC program and passes variables to it from the program currently being executed.

Remarks <filename> is the name of the program to be called by this statement. The program called will replace the current program as shown in Example 2 below. If the ALL option is specified, all variables being used by the current program are passed to the program called. If the ALL option is omitted, the calling program must contain a COMMON statement to list the variables that are to be passed. (See the discussion of the COMMON statement.)

<line number exp> is a variable or constant indicating the line number at which execution of the called program is to begin. If omitted, execution will begin with the first line of the program called. The value of this expression is not affected by execution of the RENUM command. Specifying the MERGE option makes it possible to call a subroutine into the calling program as an overlay, allowing two programs to be combined. However, if the MERGE option is specified, the called program must be an ASCII file (see the discussion of the SAVE command). When combining two programs using the MERGE option, it is usually desirable to ensure that the range of program line numbers in both programs are mutually exclusive; the reason for this is that the statements on program lines in the calling program will be replaced by statements on similarly numbered lines in the called program. Unneeded program lines in the calling program may be deleted with the DELETE option.

Note that user defined functions and variable type definitions with the DEFINT, DEFSNG, DEFDBL, or DEFSTR statements are not preserved if the MERGE option is omitted. Therefore, these statements must be restated in the chained program if it is to use the corresponding variables or functions.

See also COMMON, MERGE, SAVE

Example 1

```
Ok
150 'SAMPLE3
160 X=X^2
170 PRINT X,Y,Z
SAVE "SAMPLE3"
Ok
NEW
Ok
100 READ Y,Z
110 X=Y+Z
120 PRINT X
130 DATA 1,2
140 CHAIN "A:SAMPLE3",,ALL
RUN
 3
 9          1          2
Ok
```

Example 2

```
10 'MAIN program
20 PRINT "[ MAIN PROGRAM ]"
30 PRINT "EPSON"
40 CHAIN MERGE "SUB",10
```

```
10 'SUB program
20 '
30 PRINT "[ CHAINED PROGRAM ]"
40 PRINT "QX-10"
```

```
[ MAIN PROGRAM ]
EPSON
[ CHAINED PROGRAM ]
QX-10
Ok
```

CIRCLE

Format CIRCLE [STEP] (<horizontal position>, <vertical position>), <radius>[,[<color>][,[<starting angle>][,[<ending angle>][,<ratio>]]]]

Purpose This statement draws circles, ellipses, or arcs.

Remarks This statement draws a circle or arc around the center specified by <horizontal position> , <vertical position>; if STEP is specified, a position relative to the last coordinates used by a PRESET, PSET, CIRCLE, LINE, or CONNECT statement is assumed as the center. An arc is drawn if <starting angle> and <ending angle> are specified; these are specified in radians as values in the range from -2π to 2π, and indicate the starting position and ending position of the arc. If a negative value is specified, lines will be drawn between the center and the starting and ending positions; however, the absolute values of the numbers specified are used in determining the starting and ending positions. The default value for <starting angle> is 0, and that for <ending angle> is 2π.

<color> specifies the color to be used in drawing the circle or arc; if omitted, the current foreground color is assumed.

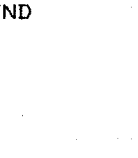
<ratio> specifies the ratio of the X diameter and the Y diameter of the circle. A circle is drawn if 1 is specified. If a value smaller than 1 is specified, <radius> becomes the X diameter; if a value larger than 1 is specified, <radius> becomes the Y diameter. The default value for <ratio> is 1.

After execution of the CIRCLE statement, the last reference pointer (LRP) is updated to <horizontal position>, <vertical position>.

See also COLOR, CONNECT, LINE, PRESET, PSET

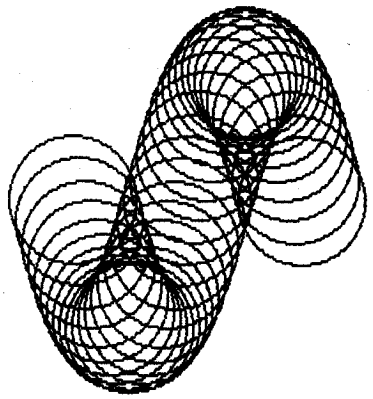
Example 1

```
10 CLS
20 CIRCLE (100,100),50
30 END

Ok
```



Example 2

```
10 CLS
20 PI=3.14159
30 D=PI/180
35 PRESET (130,180)
40 FOR I=0 TO 360 STEP 10
50 CIRCLE STEP (I/2,SIN(D*I)*100),50
60 NEXT I
70 END

Ok
```



CLEAR

Format

CLEAR [[<dummy 1>][,[<upper memory limit>][,<dummy 2>]]]

Purpose

This command clears all numeric and string variables. When <upper memory limit> is specified, it also reserves a memory area for machine language programs; this is done by setting an upper boundary in memory which defines the highest address which can be used by MFBASIC for storage of program text or variables.

Remarks

The CLEAR command cancels all variable type definitions made with the DEF statements (DEFINT, DEFSNG, DEFDBL, and DEFSTR) and closes all files which are currently open. <upper memory limit> specifies the highest address in memory which can be used by MFBASIC for storage of program text or variables. Therefore, the area from <upper memory limit> + 1 to the beginning of the stack area (see the memory map in Appendix K) can be used for storage of machine language programs. Usually, 512 bytes are reserved for the stack area; however, the size of this area will vary if a value other than 512 is specified in the /T: option when the MFBASIC command is executed. Thus, the highest address which can be used for storage of machine language programs varies accordingly.

<dummy 1> and <dummy 2> are included to provide compatibility with versions of BASIC which are used with other machines, and have no purpose in MFBASIC for the QX-10.

Example

```
10 CLEAR ,&HBFFF
```

CLOSE

Format CLOSE[[#]<file number>[,[#]<file number...>]]

Purpose This statement terminates access to device files.

Remarks <file number> is the number under which the file(s) was opened. If the CLOSE statement is executed without specifying a file number, all files currently open are closed.

Once a file has been closed, it can be reopened under a different number, or the number previously assigned to that file can be used to open a different file.

Executing this statement to close a sequential file which has been opened in the output mode causes the contents of the output buffer to be written to the end of the file.

All files are closed automatically if END, CLEAR, or NEW is executed.

See also END, OPEN, Chapter 5

CLS

Format CLS

Purpose This statement clears the display screen.

Remarks In the color mode, this statement clears the screen to the current background color; in the black and white mode, it clears the screen to black.

This statement performs the same function as PRINT CHR$ (12).

COLOR

Format COLOR [<foreground color>][,<background color>]

Purpose This statement is used to specify the foreground and background colors of the display screen.

Remarks The color specified for <foreground color> becomes the default value for displaying characters or graphics using PSET, LINE, CONNECT, and CIRCLE. The color specified for <background color> becomes the background color of the display, and is used as the default value for the PRESET statement. <foreground color> is specified as a number from 0 to 31; the meanings of the numbers from 0 to 7 are as follows.

Color mode	Black and white mode
0 ... Black	0 ... Black
1 ... Blue	1 ... White
2 ... Red	2 ... White
3 ... Violet	3 ... White
4 ... Green	4 ... White
5 ... Light blue	5 ... White
6 ... Yellow	6 ... White
7 ... White	7 ... White

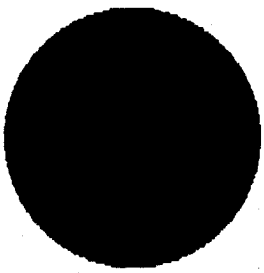
For numbers from 8 to 31, the effective color is equal to <foreground color> MOD 8); values in this range specify various screen attributes as follows.

8 to 15 ... Reverse display
16 to 23 ... Underline
24 to 31 ... Underline reverse

<background color> is specified as a number from 0 to 7; the meanings of the numbers are the same as for <foreground color>. Note that, when <background color> is changed by executing a COLOR statement, the background color does not actually change until a CLS statement is executed or the CLEAR key is pressed.
If either <background color> or <foreground color> is omitted, the value previously specified is assumed as the default value.
If the same value is specified for both <background color> and <foreground color>, characters and graphics displayed will not be visible.

Example

```
10 COLOR 2,7
20 CLS
30 CIRCLE (320,200),100
40 PAINT (320,200)
50 A$=INKEY$
60 IF A$="" THEN 50 ELSE 70
70 COLOR 7,0
80 CLS
```



COMMON

Format COMMON <list of variables>

Purpose This statement passes variables to a CHAINed program.

Remarks When one program is called by another with the CHAIN statement, the values of variables can be preserved for use by the called program by means of the COMMON statement. The COMMON statement may appear anywhere in the calling program, but the normal practice is to include it near the beginning of the program. More than one COMMON statement may be specified in a program, but the same variables cannot be specified in more than one COMMON statement. Array variables are specified by appending "( )" to the array name. (Note: If all variables are to be passed to the CHAINed program, use the CHAIN statement with the ALL option instead of the COMMON statement; see the explanation of the CHAIN statement.)

See also CHAIN

Example

```
10 PRINT "A=";A,"B=";B,"C=";C
20 FOR I=1 TO 10
30 PRINT "X(";I;")=";X(I)
40 NEXT I
SAVE "PRINT"
Ok
```

```
10 COMMON B,X()
20 A=1:B=2:C=3
30 FOR I=1 TO 10
40 X(I)=I
50 NEXT I
60 CHAIN "PRINT"
RUN
A= 0          B= 2          C= 0
X( 1 )= 1
X( 2 )= 2
X( 3 )= 3
X( 4 )= 4
X( 5 )= 5
X( 6 )= 6
X( 7 )= 7
X( 8 )= 8
X( 9 )= 9
X( 10 )= 10
Ok
```

CONNECT

Format CONNECT[STEP](X1,Y1)-[STEP](X2,Y2)[-[STEP](X3,Y3)...
-[STEP](Xn,Yn)][,[<color code>][,<line style>]]

Purpose The CONNECT statement draws lines between specified points.

Remarks This statement draws lines between the dot coordinates specified by (X1,Y1) through (Xn,Yn), starting at (X1,Y1) and proceeding in sequence to (Xn,Yn). Relative coordinates can be specified for each point by preceeding its coordinate specification with STEP.

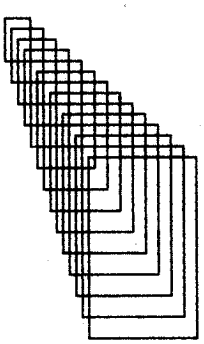
<color code> specifies the color of lines drawn; if omitted, the current foreground color is used.

The <line style> parameter specifies the style to be usd in drawing lines, and is specified as any number from &H0 to &HFFFF (see the explanation of the LINE statement for a discussion of line style specification); the default value is &HFFFF.

See also CIRCLE, COLOR, LINE, PSET, PRESET

Example

```
10 CLS
15 PSET (20,20)
20 CONNECT STEP(0,0)-STEP(0,20)-STEP(20,20)-STEP(20,0)-STEP(0,0)
25 FOR X=30 TO 90 STEP 5
30 CONNECT STEP(5,5)-STEP(5,X)-STEP(X,X)-STEP(X,5)-STEP(5,5)
40 NEXT X
RUN

Ok
```



CONT

Format CONT

Purpose The CONT command is used to resume execution of a program which has been interrupted by a STOP or END statement, or by pressing CTRL and C or the BREAK key.

Remarks This command causes program execution to be resumed at the point at which it was interrupted. If execution is interrupted during display of the prompt by an INPUT statement ("?" or a prompt string defined by the user), the prompt is displayed again when program execution is resumed.

The CONT command is often used in conjunction with the STOP statement for debugging. This is usually done by interrupting the program and executing statements in the direct mode to examine and/or change intermediate values, then resuming execution with CONT (or a direct mode GOTO statement to resume execution at a different line number). The CONT statement can also be used to resume execution of a program which has been interrupted by an error; however, it cannot be used if any changes are made in the program during the break.

Example See the example for the STOP statement.

COPY

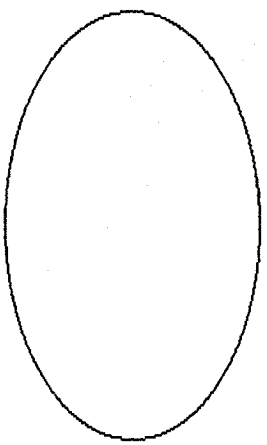
Format COPY

Purpose The COPY statement is used to output the contents of the display to the printer.

Remarks The COPY statement outputs the contents of the CRT display to the printer in bit image format. Operation is the same as when the SCRNDUMP key is pressed.

Example

```
10 CLS
20 CIRCLE (300,200),100
30 COPY
RUN
```



DATA

Format DATA <list of constants>

Purpose The DATA statement is used to store numeric and string constants which are substituted into variables by the READ statement. (See the explanation of the READ statement.)

Remarks DATA statements are non-executable, and may be located anywhere in the program. Constants included in the list must be separated from each other by commas, and are substituted into variables upon execution of READ statements in the order in which they appear in the list. A program may include any number of DATA statements.

When more than one DATA statement is included in a program, they are accessed by the READ statements in the order in which they appear (program line number order); therefore, the lists of constants specified in DATA statements can be thought of as constituting one continuous list, regardless of the number of constants on each individual line or where the lines appear in the program.

Constants of any type (numeric or string) may be included in <list of constants>; however, the types of the constants must be the same as the types of variables into which they are to be substituted by READ statements. It is not possible to combine numeric or string constants in <list of constants> with operators. Further, string constants must be enclosed in quotation marks if they include commas, colons, or significant leading or trailing spaces; otherwise, quotation marks are not required.

Once the <list of constants> of a DATA statement has been read, it cannot be read again until a RESTORE statement is executed.

See also READ, RESTORE

Example 1

```
10 READ A,B,C
20 PRINT A;B;C
30 DATA 1,2,3,4,5,6,7,8,9,10
40 READ A,B,C
50 PRINT A;B;C
60 READ A,B,C
70 PRINT A;B;C
80 END
```

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

Example 2

```
10 READ A,B,C
20 PRINT A;B;C
30 DATA 1,2,3,4,5,6,7,8,9,10
40 RESTORE
50 READ A,B,C
60 PRINT A;B;C
70 END
```

```
RUN
 1  2  3
 1  2  3
Ok
```

Example 3

```
10 READ A,B,C
20 PRINT A;B;C
30 DATA 1,2,3
40 DATA 4,5,6
50 READ A,B,C
60 PRINT A;B;C
70 RESTORE 30
80 READ A,B,C
90 PRINT A;B;C
100 RESTORE 130
110 READ A,B,C
120 PRINT A;B;C
130 DATA 7,8,9
140 END

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

DATE$

Format

As a statement
DATE$ = "<MM>/<DD>/<YY>"
As a variable
X$ = DATE$

Purpose

DATE$ is a system variable which contains the date of the QX-10's calendar clock.

Remarks

As a statement, DATE$ is used to set the date of the QX-10's built-in calendar clock. <MM> is a number from 01 to 12 which indicates the month, <DD> is a number from 01 to 31 which indicates the day, and YY is a number from 00 to 99 which indicates the year.
As a variable, DATE$ returns the date of the built-in clock in "MM/DD/YY" format.

See also

DATE, DAY

Example

```
10 INPUT "ENTER TODAY'S DATE (MM/DD/YY)";D$
20 DATE$=D$
30 INPUT "ENTER DAY OF WEEK (0=SUN,1=MON,2=TUE,3=WED
,4=THU,5=FRI,6=SAT)";D
40 DAY=D
50 CLS
60 DIM D(12)
70 FOR I=1 TO 12
80 READ D(I)
90 NEXT
100 TD=VAL(MID$(DATE$,4,2))
110 TM=VAL(LEFT$(DATE$,2))
120 TY=VAL(RIGHT$(DATE$,2))
130 IF TY MOD 4=0 AND TM=2 THEN D(2)=29
140 D=DAY
150 FOR I=1 TO TD-1
160 D=D-1:IF D<0 THEN D=6
170 NEXT
180 FOR I=1 TO TM
190 READ A$
200 NEXT
210 LOCATE 40-(LEN(A$)+5)/2
220 PRINT A$;1900+TY
230 LOCATE 6,2
240 RESTORE 450
250 FOR I=1 TO 7
260 READ A$:PRINT A$;SPC(8);
270 NEXT
280 L=3
290 FOR I=1 TO D(TM)
300 LOCATE (D MOD 7)*11+2,L
310 IF L=18 THEN LOCATE (D MOD 7)*11+10,L-1:LINE (5+
88*DI,337)-(5+88*(DI+1),277):DI=DI+1
320 IF I=TD THEN COLOR 0,7
330 PRINT MID$(STR$(I),2,2);
340 COLOR 7,0
350 D=D+1:IF D=7 THEN D=0:L=L+3
360 NEXT
370 FOR I=0 TO 5
380 LINE (5,37+60*I)-(621,37+60*I)
390 NEXT
400 FOR I=0 TO 7
410 LINE (5+88*I,37)-(5+88*I,337)
420 NEXT
430 DATA 31,28,31,30,31,30,31,31,30,31,30,31
440 DATA JANUARY,FEBRUARY,MARCH,APRIL,MAY,JUNE,JULY,
AUGUST,SEPTEMBER,OCTOBER,NOVEMBER,DECEMBER
450 DATA SUN,MON,TUE,WED,THU,FRI,SAT
460 GOTO 460
```

FEBRUARY 1983

SUN	MON	TUE	WED	THU	FRI	SAT
		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					

DAY

Format

As a variable
X% = DAY
As a statement
DAY = <W>

Purpose

DAY is a system variable which maintains the day of the week of the QX-10's calendar clock.

Remarks

As a variable, DAY returns the day of the week of the QX-10's built-in clock as a number from 0 to 6. Normally, "0" is used for Sunday and "6" is used for Saturday.

However, the day can be set independently of the value assigned as the calendar date (by the DATE$ statement); therefore, as a statement, DAY can be used to assign any number from 0 to 6 to the current day of the week.

Example

```
10 'THIS PROGRAM SETS THE SYSTEM'S CALENDAR, COMPUTES
20 'THE NUMBER OF DAYS BETWEEN DATES, AND COMPUTES THE
30 'DAY OF THE WEEK FOR A GIVEN DATE.
40 INPUT"ENTER TODAY'S DATE(MM/DD/YY)";A$
50 DATE$=A$
60 INPUT"ENTER DAY OF WEEK (SUN=0,MON=1,TUE=2,WED=3,THU
=4,FRI=5,SAT=6)";A
70 DAY=A
80 CLS
90 W$="SUNMONTUEWEDTHUFRISAT"
100 PRINT "TODAY IS ";MID$(W$,DAY*3+1,3);", ";DATE$
110 M1=VAL(LEFT$(DATE$,2)):D1=VAL(MID$(DATE$,4,2)):Y1=1
900+VAL(RIGHT$(DATE$,2))
120 YY=Y1:DD=D1:MM=M1
130 IF M1<3 THEN GOSUB 230 ELSE GOSUB 250
140 F1=X
150 INPUT"ENTER PAST OR FUTURE DATE(MM/DD/YYYY)";A$
160 M2=VAL(LEFT$(A$,2)):D2=VAL(MID$(A$,4,2)):Y2=VAL(RIG
HT$(A$,4))
170 YY=Y2:DD=D2:MM=M2
180 IF M2<3 THEN GOSUB 230 ELSE GOSUB 250
190 F2=X
200 PRINT "DAY IS ";MID$(W$,3*(6+F2+INT(-F2/7)*7)+1,3)
210 PRINT"NO. OF DAYS BETWEEN DATES IS";ABS(F2-F1)
220 PRINT:PRINT"HIT ANY KEY TO CONTINUE":A$=INPUT$(1):G
OTO 80
230 X=365*YY+DD+31*(MM-1)+INT((YY-1)/4)-INT(3/4*INT(((Y
Y-1)/100)+1))
240 RETURN
250 X=365*YY+DD+31*(MM-1)-INT(.4*MM+2.3)+INT(YY/4)-INT(
3/4*(INT(YY/100)+1))
260 RETURN
```

```
TODAY IS MON, 02/28/83
ENTER PAST OR FUTURE DATE(MM/DD/YYYY)? 05/08/1950
DAY IS MON
NO. OF DAYS BETWEEN DATES IS 11984

HIT ANY KEY TO CONTINUE
```

DEF FN

Format DEF FN<name> (<parameter list>) = <function definition>

Purpose The DEF FN statement is used to define and name user-written functions.

Remarks A user defined function is a numeric or string expression which expands the application range of MFBASIC. When such a function is called, the variables specified as its arguments (either in the function definition or in the parameter list of the calling statement) are substituted into the expression and the equivalent value is returned as the result of the function.

<parameter list> comprises those variables in the function definition that are to be replaced when the function is called. The items in the list are separated by commas.

If a <parameter list> is included in the <function definition>, then a list with a corresponding number of parameters must be specified in the statement calling the function; the values of variables specified in the calling statement's parameter list are then substituted into the <parameter list> of the function definition on a one-to-one basis.

<function definition> is an expression that performs the operation of the function. It is limited to one line.
Variable names that appear in this expression serve only to define the function; they do not affect program variables that have the same name.

A variable name used in a function definition may or may not appear in the parameter list. If it does, the value of the parameter is supplied when the function is called. Otherwise, the current value of the variable is used.

If a type is specified in the function name, the value of the expression is forced to that type before it is returned to the calling statement. If a type is specified in the function name and the argument type does not match, a "Type mismatch" error occurs.

The DEF FN statement must be executed before the corresponding user function can be called; otherwise, an "Undefined user function" error will occur.

DEF FN statements cannot be executed in the direct mode.

Example 1

```
10 DEF FNA(X,Y)=X*3/(Y+2)
.
.
100 R=FNA(I,J)
.
.
```

Line 10 defines function FNA. Line 100 calls the function and substitutes variables I and J for X and Y as its arguments.

Example 2

```
10 DEF FNA(X,Y)=X*Y
20 INPUT"ENTER A";A
30 INPUT"ENTER B";B
40 PRINT FNA(A,B)
50 END

RUN
ENTER A? 3
ENTER B? 7
 21
Ok
```

DEFINT/SNG/DBL/STR

Format

DEF | INT | SNG | DBL | STR | <range(s) of letters>

Purpose

This statement declares the type of variables specified in <range(s) of letters>.

Remarks

This statement defines the type of the specified variable or range of variables, making it unnecessary to indicate their type by appending the symbols %, !, #, or $. Type declarations made using this statement apply to all variable names which begin with the letters included in <range(s) of letters>. For example, execution of DEFSTR A-C declares all variables whose names begin with the letters A, B, and C as string variables even though the declaration character $ is not appended to their names. The variable type declarations specified in DEF statements do not become effective until those DEF statements have been executed; therefore, MFBASIC assumes that all variables without type declaration characters are single precision variables until a type definition statement is encountered (variables without type definition characters are then cleared when a DEF statement specifying the first letter of their names is executed).

Example 1

```
10 DEFDBL L-P
```

Declares all variables whose names begin with the letters L, M, N, O, and P as double precision variables.

Example 2

```
10 DEFSTR A
```

Declares all variables whose names begin with the letter A as string variables.

Example 3

```
DEFINT I-N,W-Z
```

Declares all variables whose names begin with the letters I to N and W to Z as integer variables.

DEF USR

Format DEF USR[<digit>]=<integer expression>

Purpose The DEF USR statement is used to specify the starting address of a user-written machine language program.

Remarks Machine language programs whose starting addresses are defined with the DEF USR statement can be used as functions in MFBASIC programs (by means of the USR function; see the explanation of the USR function and Appendix E for further information). <digit> is a number from 0 to 9 by which the machine language program is identified when called with USR; if <digit> is not specified, 0 is assumed.

<integer expression> is the starting address of the machine language program. Up to 10 starting addresses (USR0 to USR9) may be concurrently defined; if more addresses are required, additional DEF USR statements may be executed to redefine starting addresses for any of USR0 to USR9.

Machine language programs used as subroutines by MFBASIC programs must be written into memory before they can be called; the top address of the area into which machine language programs are written must be specified with the CLEAR statement.

See also CALL, USR, Appendix E

Example

```
10 CLEAR ,&HBFFF
20 ADRS=&HC000
30 FOR I=1 TO 6
40 READ A
50 POKE ADRS,A
60 ADRS=ADRS+1
70 NEXT I
80 DATA &H0E,&H01,&HCD,&H39,&HF6,&HC9
90 DEF USR2=&HC000
100 INPUT "INPUT NUMBER FROM 1 TO 254";B
110 POKE &HC001,B
120 A=USR2(A)
130 GOTO 100

RUN
INPUT NUMBER FROM 1 TO 254? 5
INPUT NUMBER FROM 1 TO 254? 70
INPUT NUMBER FROM 1 TO 254?
```

DELETE

Format DELETE [<line number 1>][—<line number 2>]

Purpose Execution of the DELETE command deletes specified program lines.

Remarks If both <line number 1> and —<line number 2> are specified, all program lines from <line number 1> to <line number 2> will be deleted. Only the line specified in <line number 1> will be deleted if the second parameter is omitted; if the first parameter is omitted, all lines from the beginning of the program to <line number 2> will be omitted. Note that MFBASIC always returns to the command level after execution of a DELETE statement, and that an "Illegal function call" error will result if a specified line number does not exist.

Example 1 DELETE 40
Deletes program line 40.

Example 2 DELETE 40-100
Deletes all program lines from 40 to 100.

Example 3 DELETE -40
Deletes all lines from the beginning of the program to line 40.

DIM

Format DIM <list of subscripted variables>

Purpose The DIM statement is used to specify the maximum range of array subscripts and to allocate space for storage of array variables.

Remarks The DIM statement defines the extent of each dimension of variable arrays by specifying the maximum value which can be used as subscripts for each dimension; it also clears all variables in the specified array(s). For example, DIM A(25,50) defines a two-dimensional array whose individual variables are designated as A(N1,N2), where the maximum value of Nl is 50 and the maximum value of N2 is 25. Since the minimum value of a subscript is 0 (unless otherwise specified with the OPTION BASE statement), this array includes $51 \times 26 = 1326$ individual variables. Any attempt to access an array element with subscripts greater than those specified in the DIM statement for that array will result in a "Subscript out of range" error; if no DIM statement is specified, the maximum value which can be used for subscripts is 10. Once an array has been dimensioned with the DIM statement, it cannot be redimensioned until it has been erased by a CLEAR or ERASE statement.

See also ERASE, OPTION BASE, CLEAR

Example 1
```
10 DIM A(20,15)
```
Defines two-dimensional array A and specifies 20 and 15 as its maximum subscript values. Unless otherwise specified by a DEF<type> statement, MFBASIC will handle this as a single precision numeric array.

Example 2
```
10 DIM A$(30)
```
Defines one-dimensional string array A$ and specifies 30 as its maximum subscript value.

Example 3
```
10 DIM G%(25),F%(25)
```
Defines one-dimensional arrays G and F and specifies 25 as the maximum values of their subscripts.

EDIT

Format EDIT <line number>

Purpose This command instructs MFBASIC to enter the EDIT mode for the specified line.

Remarks When this command is executed, MFBASIC displays the program line specified by <line number>, positions the cursor to the first character in that line, and waits for changes to be made. Changes are made using the screen editing keys as described in section 1.6. After a line has been changed on the screen, it is stored in memory and the EDIT mode is exited by pressing the RETURN key. The former contents of the line will not be changed if BREAK or CTRL and C is pressed instead of the RETURN key.

NOTE:

If a syntax error is encountered during program execution, MFBASIC automatically enters the EDIT mode at the line containing the error. The line can then be edited as described above; however, all variables will be cleared when the RETURN key is pressed. If you wish to examine the contents of variables, exit the EDIT mode by pressing the BREAK key or CTRL and C.

END

Format END

Purpose When encountered during program execution, this statement terminates execution, closes all files, and returns MFBASIC to the command level.

Remarks END statements may be included in a program at any point desired. Unlike the STOP statement, END does not cause a BREAK message to be displayed. The END statement is optional when the last program line of a program is the last line executed.

Example 1

```
10 A=-2
20 B=-3
30 C=A*B
40 GOTO 60
50 END
60 PRINT "A=";A,"B=";B,"C=";C
70 GOTO 50

RUN
A=-2          B=-3          C= 6
Ok
```

Example 2 When the END statement is encountered before the last program line, execution can be resumed with the CONT command.

```
10 PRINT "EPSON"
20 END
30 PRINT "QX-10"
Ok
RUN
EPSON
Ok
CONT
QX-10
Ok
```