ENHBAS

ENHBAS version 2.X is a great improvement over earlier version of ENHBAS. Instead of the interpretation scheme of the "old" ENHBAS, version 2.X actually tokenizes ENHBAS reserved words, just as Level II BASIC tokenizes words like PRINT, INPUT, etc. Therefore, execution time is faster and program text is more compressed.

In addition, several commands have been added since version 1.6, including the PLAY command which allows you to play a series of musical notes over the cassette output port (described in the latest ENHBAS manual.) Also, two new commands have been added since version 1.8: ZSTEP and OUTPUT. They are described on page 28.1, which is included with this addendum sheet.

Several other changes have occurred. Instead of using the down arrow as a control character, <SHIFT> <CLEAR> now performs the special control key function for shorthand entries. Shorthand entry "0" no longer functions.

Because of a delay problem with the Epson MX-80 printer, ENHBAS now delays for approximately one minute before issuing a PRINT NOT READY error.

Important: Any programs written under the "old" ENHBAS will not run unmodified under post 2.0 ENHBAS, because of the reserved word tokenizing. To convert these programs, you must first save them out as an ASCII file and then re-load them to automatically convert the program, with a disk system. With a tape system, you must load the old ENHBAS program under the new ENHBAS, and then EDIT any line that has an ENHBAS word in it, such as "LEFT" or "SORT". It is not necessary to mkae any changes in the line, simply hit enter after going into the edit. This will automatically tokenize any ENHBAS reserved words in that line. After editting all relevant lines, resave the program out to tape. This procedure is necessary only once.

The commands ROT and SCALE have been added to Model I ENHBAS, and it is recommended that you program with these commands instead of using the POKE locations supplied in the documentation for the DRAW command. ROT = exp, and SCALE = exp, eliminate the need to POKE 16426, exp or POKE 16427,exp, and provide compatibility with the Model III version, where the Model I POKE locations are invalid.

Table of Contents

Table of Contents (con't)

NEW FUNCTIONS

LOADING ENHBAS

For Model I disk

ENHBAS is supplied on a serialized "transfer" disk that has no operating system on it. It merely transfers the correct version of ENHBAS to one of YOUR system disks.

To use, simply:

1) Insert the "transfer" disk into your disk drive.
2) Boot-up by pressing the reset button at the back of your machine. A special utility program will be loaded along with the proper version of ENHBAS (determined by the memory size of the machine being used)
3) When prompted, remove the "transfer" disk and insert a TRSDOS or NEWDOS disk with at least 4 free GRANS. Then hit <ENTER>. When your disk boots up, ENHBAS will be dumped onto your disk.
4) Once the dump is complete, you will have a working copy of ENHBAS on your system disk. Since ENHBAS automatically loads BASIC, make sure the disk has disk BASIC, and that is named BASIC.

* If you have NEWDOS-80 (model I), you must first dump ENHBAS onto a TRSDOS or NEWDOS 1.1 or 1.2 disk. Then, remove the disk and boot up with a NEWDOS-80 system disk. Then, copy ENHBAS over to your NEWDOS-80 disk with:

```
COPY :0 $ENHBAS/CMD ENHBAS/CMD
```

5) Now, instead of typing BASIC as you would normally, type ENHBAS. You may include any parameters after the name ENHBAS, if necessary, as with NEWDOS, just as you would after the name BASIC.

IMPORTANT: See note under Model III loading instructions

32K Disk ENHBAS loads from AD00-BF00; 48K from ED00-FF00.

For Model I TAPE

Loading tape ENHBAS is easy.

1) Place either side of the cassette into the recorder and rewind to the beginning.
2) Set volume controls at 5-6
3) Type SYSTEM <ENTER>
4) Type ENHBAS <ENTER> (be careful not to transpose N,H)
5) Press the PLAY button on your recorder
6) Make sure the earphone jack is connected to the player
7) ENHBAS should begin to load
8) Blinking asterisks should appear in the upper right hand corner of the screen
9) If a 'C' appears, turn the recorder off and rewind the tape to the beginning. Try a higher volume level
10) If the asterisks come on but do not blink for over a minute, stop the recorder and rewind the tape to the beginning, and try a lower volume level
11) If you have persistent problems, clean your tape player
12) If all goes well, ENHBAS will self-execute

Tape ENHBAS loads in lower memory from 42E9 to about 5500.

For Model III disk

Because the vast majority of Model III disk users have two drives, ENHBAS is supplied on a formatted disk. There are two separate versions of ENHBAS, a 32K and 48K version. To copy the appropriate version of ENHBAS over to your system disk, merely place the formatted disk in drive 1 and your system disk in drive 0. Then, type in:

```
COPY ENH32/CMD:1 ENHBAS/CMD:0          for a 32K machine
COPY ENH48/CMD:1 ENHBAS/CMD:0          for a 48K machine
```

You must have at least 8 grans free on your system disk. Once the transfer is complete, you will have a working copy of ENHBAS on your disk. You may, of course, want both versions of ENHBAS on your disk. In that case, you should probably retain the same names that are on the formatted disk.

IMPORTANT: If you go to DOS from ENHBAS, do NOT retype ENHBAS to go back. Type BASIC. ENHBAS will still be resident, unless you overlaid it with some other program while you were in DOS; then, you should re-boot the system before typing ENHBAS.

Model III disk ENHBAS loads at the same addresses as the Model I.

For Model III tape

Model III tape loading instructions are identical to Model I tape loading instructions, on the flip side of this page. Note that ENHBAS is recorded at the "old" speed of 500 baud, not the "new" speed of 1500 baud. This is because the higher baud rate loads unreliably with many tape recorders.

Model III tape ENHBAS loads from 4300 to about 5800 (hex).

Introduction

Welcome to ENHBAS (pronounced "EN"-"BASE"). With this one purchase you have expanded your BASIC language and your Model II with the most useful and comprehensive commands and utilities available anywhere at any price:

FAST SORTS	FORMATTED LISTING
CALCULATED BRANCHING	LINE LABELS
"EVAL"UATED STRINGS	RE"NEW" PROGRAMS
WHILE/WEND LOOPING	"RESTORE" AT ANY LINE #
SCROLL PROTECT	SOUND UTILITIES
USER-DEFINED CURSORS	ONE-LETTER COMMANDS
and many others	

Some of the features of ENHBAS would cost hundreds of dollars if bought in bits and pieces from other companies, and many ENHBAS commands are not available ANYWHERE. There are imitations of ENHBAS, but none even remotely compare to the real article.

The Cornsoft Group provides no warranty express or implied on the operation of ENHBAS, and assumes no responsibility for any damages resulting from the direct or indirect use of ENHBAS, including but not limited to loss of revenue from errors imagined or real in ENHBAS.

Further, The Cornsoft Group assumes no responsibility for errors or omissions in this document. The Cornsoft Group assumes no responsibility for any damages resulting from the use of information supplied herein.

ENHBAS is copyrighted by The Cornsoft Group, and a notarized copy of the source code is on file. You have permission to make copies of ENHBAS or any derivatives of ENHBAS thereof only for your own personal use. It is illegal to make copies of ENHBAS for anyone but yourself. Please note that all copies of ENHBAS are serialized, both obviously in the form of an ASCII message, and in subtler ways. We also keep track of the names of individuals coupled with their serial #'s, so please think twice about making a copy for someone who may in turn make 100 copies ....

It is advisable to store the supplied diskette or tape in an environmentally safe place, away from power supplies and other sources of magnetic fields, and extremes of humidity and temperature.

SPECIAL FEATURES

ONE-LETTER COMMANDS

Typing in LIST, EDIT, and so on becomes tedious after a while. Therefore, ENHBAS provides shorthand equivalents of these lengthy but often used commands:

Abbreviation	EQUIV. command
D	DELETE
E	EDIT
F	FIND
I	AUTO
L	LIST
P	LIST.-
R	RUN

FORMATTED LISTING

Both screen and printer listings in ENHBAS are now formatted for easier reading. All program lines are neatly columnized at position 6. For example:

```
10    '  THIS PROGRAM SHOWS HOW THE NEW PROGRAM LISTING WORKS. WHENEV
      ER A PROGRAM LINE OVERFLOWS A LINE, IT IS AUTOMATICALLY WRAPPED
      AROUND AND "TAB"BED OVER 6 SPACES.
1000  '  SINCE THE LISTING IS TABBED 6 SPACES, ALL LINES, REGARDLESS
      OF THE NUMBER OF DIGITS IN THE LINES NUMBER, LINE UP UNDER EACH
      OTHER.
10000 '  OBVIOUSLY, THIS PROGRAM LISTING IS MUCH EASIER TO READ BECA
      USE OF THIS ENHBAS UTILITY.
```

CONTROL KEY

All non-Model II versions of ENHBAS alter the CLEAR key to act as a conventional control key. For instance, hitting CLEAR and C at the same time will produce the same effects as BREAK. Most of the standard control key functions are provided for:

Control-A	Like "normal" BREAK (no screen echo)
Control-C	Like BREAK, except echos to video screen
Control-G	Bell
Control-H	Backspace
Control-I	Tab
Control-J	Linefeed

All control-letters, except control-A and control-C, return ASCII codes from 2 to 26 (i.e. control-B returns a 2.)

The control key is also used as follows:

Control-1 thru Control-6
With the lower-case modification, control-1 thru 6 yield the "special" characters, such as brackets.

Control-dash ("minus-sign")
Makes an underline character

Control-arrows (except up arrow)
Prints the arrow characters

Control-@
Clears screen and input (like "normal" CLEAR)

LOWER CASE

ENHBAS tests to see if lower-case is present, and if so, alters the I/O drivers to handle lower case. This is true for both disk and tape versions of ENHBAS. (It is unnecessary in Model III ENHBAS.)

If you have the "Electric Pencil" modification, you will need to switch to lower-case before hitting <ENTER> from DOS READY, in order to enable lower case so ENHBAS can detect it. Also, if you are running under NEWDOS, you probably know that you can't use lower-case in a BASIC program, because the disk BASIC converts all lower-case to upper-case automatically (Newdos +, NOT Newdos-80). You can disable this with:

```
POKE 20992,201
```

entered from BASIC.

Control-0 (zero, not the letter O) toggles the keyboard from the "normal" mode (<SHIFT> for lower-case) to the "typewriter" mode (<SHIFT> for upper-case).

SOUND

ENHBAS contains several utilities which involve sound. In order to hain full use of these utilities you will need to connect a small amplifier/speaker (similar to Radio Shack Cat. No. 277-1008) to the computer's cassette output jack.

— <BREAK> or (control-C) makes a distinctive, high-pitched metallic tone

— Any error causes a short two-tone beep

— A "click" is generated any time a key is depressed. This key-click allows some people to type faster because of the audio feedback. Key-click is initially enabled. To disable:

```
        POKE 16408,1
```

and to enable:

```
        POKE 16408,0
```

Note that all control characters still make a click, even when key-click is disabled

— Control-G (BELL) is supplied disabled. To enable:

```
        POKE 16409,1
```

To disable:

```
        POKE 16409,0
```

Whenever a control-G is encountered, a BELL (in this case, Westminster Chimes) is generated.

The PLAY command (described in detail elsewhere in this manual) gives you full control of the ENHBAS sound playing routines.

LINE-PRINTER SELECT (Model I version only)

ENHBAS will generate an error (PRINTER NOT READY) if you attempt to LPRINT or LLIST when no printer is available. It will wait for about 6 seconds before generating the message. This is great for disk users because LPRINT will no longer "hang" the system if no printer is connected.

USER-DEFINABLE CURSOR (Model I version only)

The cursor character is now stored at memory location 16419. To change this character, which is initially a block (chr$ code 143), just POKE the ASCII value of the new cursor into 16419.

SHORT-ENTRY COMMANDS

Thirty-five shorthand commands may be entered by pressing <CLEAR> and <SHIFT> keys at the same time, and any key "A"-"Z" or "0"-"9". When you do this, an entire command is entered, this saving keystrokes. The table that follows shows each command and its associated key:

KEY	COMMAND	KEY	COMMAND	KEY	COMMAND
A	AUTO	M	MID$(	Y	RIGHT$
B	BIN$(	N	NEXT	Z	CHR$(
C	CSUB	O	WHILE		
D	DIM	P	PRINT	1	LINEINPUT
E	ELSE	Q	WEND	2	OPEN
F	FOR	R	ROGOTO	3	CLOSE
G	GOSUB	S	STRING$(	4	FIELD
H	undefined	T	THEN	5	LSET
I	INPUT	U	RETURN	6	RSET
J	JNAME"	V	RESET(	7	MKD$(
K	KEY	W	WINKEY$	8	MKI$(
L	LEFT$(	X	(0)	9	MKS$(

NEW COMMANDS

INTRODUCTION

All program words in ENHBAS fall under two basic catagories: commands and functions. Commands are entered directly, such as PRINT and INPUT. Functions are in expressions such as X*EVAL(G$) or EN*Y/X.

Notice that some commands and functions listed were added to ENHBAS in a later version release. A comment is listed with the command if it was added after ENHBAS V1.6, and in what version # it first exists in.

Do not use commands as functions and functions as commands. For instance, some people have tried to use the CALL function as a direct command, when in fact it is used inside a numeric expression. Similarly, some people have tried to use EXEC inside a numeric expression and gotten a syntax error.

COMMANDS ENTERED UNDER "READY"

FIND (Find a line-label used in the program)

```
FIND  <label>
      where <label> is a string expression defining a line
      label flagged by JNAME in a program. <label> can be
      either a string literal or a string expression
```

This command searches for the first occurance of a specified line label in a program, listing the line if the label is found, or generating a LABEL NOT FOUND error if not.

EXAMPLE

```
10    JNAME"LABEL #1"
20    PRINT"Hi!"
30    END
40    JNAME"SUBROUTINE #1"
50    PRINT"Bye!"
60    RETURN
```

If the command:

```
FIND "SUBROUTINE #1"
```

was entered, the following would be printed on the screen:

```
40     JNAME"SUBROUTINE #1"
```

Note that 'F' (without quotes) can be used as a shorthand for FIND in READY mode, if 'F' is the first character in the line.

RENEW (Retrieve a "NEW"ed program)

```
RENEW
      no operand accepted
```

You can enter RENEW immediately after entering NEW to recall your program. Under some circumstances, RENEW will also recall a program if you go to TRSDOS and return, as long as you allocate the same number of files as you did previously.

SORTING COMMANDS

KEY / TAG (Set up arrays for SORT)

```
KEY <arrayname> (,<arrayname>, ...)
TAG <arrayname> (,<arrayname>, ...)
        where <arrayname> are array variable names to be
        operated on by SORT. Keying and Tagging are from
        LEAST to MOST significant order if more than one
        arrayname is specified
```

In ENHBAS, sorts are handled through numeric and string arrays. These arrays must be SINGLY dimensioned, such as A(9) or BG$(45). They may not be doubly or triply dimensioned, such as K(4,5) or J$(1,2,3)

To take a simple example, suppose the following array variables held (assume A() is dimensioned at 3):

```
A(0)=4          A(1)=1          A(2)=9          A(3)=3
```

Then, it's a simple matter to sort these numbers:

```
SCLEAR: KEY A: SORT
```

After the previous line is executed, A() would hold:

```
A(0)=1          A(1)=3          A(2)=4          A(3)=9
```

Many times you'll want to sort two or more arrays simultaneously. For example, if you had a mailing list and you were sorting the list by names, there might be an occasion when there would be two or more identical names. In such a case, you would want to sort those entries by street addresses or cities.

Let's say the arrays NL$, NF$, AD$, and ZP, hold, respectively, a list of last names, first names, street addresses, and zip codes (numeric), say for a small mailing list.

Further, let's say that you want to sort this list by two keys: last names, and if there are duplicate last names, first names. The addresses and zip codes should just "tag" along with their respective names.

The procedure is relatively easy:

```
SCLEAR: KEY NF$, NL$: TAG AD$,ZP: SORT
```

The SCLEAR initializes KEY and TAG. KEY NF$,NL$ makes array NL$ (last names) as the primary sorting key, and array NF$ (first names) as the secondary sorting key, in case there are last name duplicates such as "JONES". TAG AD$,ZP tells the SORT command to "tag" arrays AD$ and ZP (address and zip code) along with the names as they sort.

Otherwise, after a sort on the names, the address and zip code arrays would no longer match up to the proper first and last name arrays.

Notice that the KEY NF$,NL$ specified sorting keys in LEAST precedence to GREATEST precedence (NL$ as the key with greatest precedence, or the primary key, and NF$ as the key with lesser precedence.) This rule applies to any number of multiple keys.

Please note that a maximum of 14 keys and 14 tags are allowed. If this limit is exceeded, a KEY STACK FULL or TAG STACK FULL error will be issued. Note that any combination of integer, single precision, double precision, or string arrays may be specified in a multiple KEY or TAG — as long as the same array isn't specified twice.

SORT takes the arrays specified by KEY and TAG and sorts them in the specified order. There is a more detailed description of the SORT command on the next page.

IMPORTANT: all KEYed and TAGed arrays must be equally dimensioned, or else an error message will be printed when you try to SORT. Occasionally, you may want to sort up to a top limit in an array. The ATOP command, described on the next page, provides this capability. However, regardless of the ATOP top limit, the equal dimension restriction still applies.

SORT (Sort all KEYed and TAGed arrays)

```
SORT ((flag))
     where (flag) is a numerical expression that determines
     whether the sort will be ascending or descending.
SORT (0)  (or SORT)  =  Normal sort (small to large)
SORT (1)  =  Reverse sort (large to small)
```

All array items that have been KEYed and/or TAGed are sorted in the specified order (ascending or descending, see box), as discussed under KEY / TAG. Array variable types are determined strictly at KEY and TAG time, not SORT time. The ENHBAS command SCLEAR must be used before KEYing and TAGing, and setting ATOP (see below.)

The sort uses a modified Shell/Metzner technique.

EXAMPLE

```
10    ' THIS EXAMPLE WILL SIMULATE A MAILING LIST SORT.
20    '   NL$()  is the LAST NAME array
30    '   NF$()  is the FIRST NAME array
40    '   AD$()  is the STREET ADDRESS array
50    '   ZP()   is the ZIP CODE numeric array
60    '
70    SCLEAR:              ' INITIALIZE KEY/TAG
80    KEY ZP,NF$,NL$:      ' KEY ON LAST NAMES PRIMARILY,
90    '                        DEFAULT TO FIRST NAMES IF
100   '                        DUPLICATE FIRST NAMES, AND
110   '                        DEFAULT TO ZIP CODE KEY IF
120   '                        ENTIRE NAME IS THE SAME
130   TAG AD$:             ' TAG STREET ADDRESS
140   '
150   SORT:                ' SORT'EM!
```

Note that, with a small re-ordering of line 80, you can just as easily sort with zip codes as the primary key (KEY NF$,NL$,ZP).

ATOP (Sets top limit for SORT)

```
ATOP = nmexp
     where nmexp is a numerical expression denoting top
     sort limit
```

If you desire to sort only up to a certain limit in an array, set the limit with ATOP. The limit must be less than the DIM of the array(s) KEYed and TAGed for the SORT.

BRANCHING COMMANDS

JNAME (Label a program line)

```
JNAME <label>
     where <label> is any string expression, expressed as
     a string literal in quotes
```

JNAME allows you to refer to program lines by names, in addition to line numbers, by executing the ENHBAS commands GTO and CSUB (special GOTO and GOSUB.) A JNAME statement must be the first statement on a program line (in other words, it cannot be imbedded in a multiple statement line.)

EXAMPLE

```
10      INPUT A
20      GTO "PROCESS INPUT"
30      REM  NOTE THAT LINE 20 IS THE SAME AS 'GOTO 50'
40      END
50      JNAME "PROCESS INPUT"
60       A=A+1
70       PRINT A
```

The following would also have been valid for line 50:

```
50      JNAME "PROCESS INPUT": A=A+1: PRINT A
```

An UNDEFINED LABEL error will occur if a referenced label does not exist in a program.

Model I and III ENHBAS does not allow you to use ON x GTO, or ON x CSUB.

To locate labels in a program, use the FIND command.

GTO / CSUB (Special GOTO / GOSUB)

```
GTO <label>
CSUB <label>
     where <label> is a string expression representing a
     line label defined by JNAME. <label> may be either
     a string literal in quotes or a string expression

GTO <nmexp>
CSUB <nmexp>
     where <nmexp> is a numerical expression
```

GTO <label> and CSUB <label> have already been discussed on the previous page, under JNAME.

When a numeric, not a string, expression follows a GTO or CSUB, then a so called "calculated branch" is done. GTO 100+X jumps to the line # specified by 100+X (which could be 150, 600, etc.)

<u>EXAMPLE</u>

```
10    DEFINT X
20    INPUT X
30    IF (X>8) OR (X<1) THEN 20
40    GTO X*100
50    '
60    ' LINE 40 REPLACES THE MUCH LONGER:
70    '    ON X GOTO 100,200,300,400,500,600,700,800
```

An Undefined Line error will be issued if a non-existent line is referenced.

POP (Remove last GOSUB/CSUB)

```
POP
     no operand allowed
```

POP simply removes the last GOSUB or CSUB on the stack. For instance:

```
10   GOSUB 20: PRINT "line 10"
15   END
20   GOSUB 30: PRINT "line 20"
25   RETURN
30   PRINT "line 30"
40   POP: RETURN
```

If line 40 was simply "RETURN", then:

```
line 30
line 20
line 10
```

would be printed on the screen. However, since the POP at line 40 removes the GOSUB at line 20, the following will be printed:

```
line 30
line 10
```

Instead of returning to the next statement on line 20, the RETURN on line 40 went to the next statement on line 10.

RDGOTO / RDGTO (RESTORE at any given line or label)

RDGOTO <nmexp>
where <nmexp> is a numerical expression denoting a line number

RDGTO <label>
where <label> is a string expression representing a line label defined by JNAME. <label> may be either a string literal enclosed in quotes or a string variable expression

RDGOTO and RDGTO allow you to RESTORE to a data statement at any line, instead of being forced to RESTORE to the first data statement. The only distinction between RDGOTO and RDGTO is that the former restores to any given line #, say line 5000. Essentially, anything following a GTO expression can follow a RDGTO. Therefore, RDGTO allows you to use line labels to mark groups of data statements and thus provides you with the ability to seriously use DATA statements, unlike normal BASIC. Note that line numbers following RDGOTO automatically renumber under RENUM.

EXAMPLE

```
5      DATA 0,.5
10      JNAME "DATA1"
20     DATA 1, 2, 3, 4, 5
30      JNAME "DATA2"
40     DATA 6, 7, 8, 9, 10
50     '
60     FOR X=1 TO 12
70      READ A
80     NEXT
90     ' AT THIS POINT, ALL DATA ITEMS HAVE BEEN READ. BUT ...
100    RDGOTO 40
110    READ A: PRINT A
120    ' LINE 110 prints a "6"
130    RDGTO "DATA1"
140    READ A: PRINT A
150    ' AND line 140 prints a "1"
```

In the example, line 100 causes the next READ to start at line 40, using RDGOTO. Line 130 causes the next READ to start after label "DATA1", or line 10.

IMPORTANT: a DATA statement can not be used on the same line after a JNAME.

WHILE / WEND (Structured programming loop construct)

```
WHILE <test condition>;
          (program code)
WEND

          where <test condition> is a logical or relational
          to be tested for TRUE (non-zero) or FALSE (zero)
          <Test condition> may be anything allowed after an IF
```

WHILE/WEND statements allow you to loop without using GOTOs, thereby making line # independent subroutines easier to write.

The process is fairly easy. You set up a test condition after the WHILE command, and place a semicolon after it, and a standard colon after that if you plan to use a multi-statement program line. You then place whatever program code you wish after the WHILE, to be executed repeatedly as long as the condition after the WHILE is TRUE (non-zero.) Just as NEXT "closes the loop" with FOR, WEND "closes the loop" with WHILE. When a WEND is encountered, its WHILE condition is tested, and if the condition is true, program execution starts at the instruction following the WHILE. If the condition is false, program execution starts at the next statement following the WEND.

WEND WITHOUT WHILE will be printed as an error message if you execute WEND without previously executing a WHILE. WHILE/WEND statements may be nested, with somewhat the same logic as FOR/NEXT loops.

Here are two equivalent programs to demonstrate WHILE/WEND:

```
——————— EXAMPLE A ———————          ——————— EXAMPLE B ———————

10    CLS                          10    CLS
20    X = 1                        20    X = 1
30    WHILE X < 10;                30    PRINT X;
40    PRINT X;                     40    X = X + 1
50    X = X + 1                    50    IF X<10 THEN 30
60    WEND
                                   60    . . .
70    . . .
```

Notice that, as with FOR/NEXT, the code intervening WHILE and WEND will be executed once before an actual test takes place.

GRAPHICS / SCREEN RELATED COMMANDS

DRAW (uses an integer array to draw turtle graphics)

```
DRAW <flag>, @ x,y USING <intarray>
DRAW <flag>, @ x,y USING LIST <nmexp>
     where          <flag> is word designating action: SET
                           or RESET
                      x,y  is a graphics point where the DRAW
                           starts
                <intarray> is an integer array name
     and           <nmexp> is an integer numerical expression
                           defining a memory location
```

DRAW uses a series of codes contained within any selected integer array to draw a corresponding series of graphics lines on the screen. The value of the codes determines the direction and length of the lines; the direction being in 45 degree increments and the length in integer increments. The lines are usually drawn from point to point, with the endpoint of the last line being the starting point of the next.

The series of codes can be called a "shape table" since they define a shape by their order. The shape table can be contained within an integer array, or POKEed into memory. In either case, each entry consists of two bytes. The first byte determines direction (from 1 to 8, with 1 designating "north", 2 designating "northeast", 3 designating "east", and so on). The second determines the number of times to draw in the specified direction (2 steps, 5 steps, etc.) A direction code of 0 stops the DRAW command and causes the next instruction after the DRAW to be executed.

To compute the graphics code for an integer array entry, use the following formula:

array(x) = direction code + 256 * (number of times to repeat)

Whether a specified figure is SET or RESET on the screen depends on the <flag>, as shown in the box explanation. For instance, let's say you want to SET an octogon on the screen, with one corner at graphics coordinates 64,23. The following program does this:

```
10    DEFINT A
15    K = 7 * 256:         'To repeat each code 7 times
20    A(0) = 1 + K         'To draw a line (7 blocks long) north
                           'from 64,23
30    A(1) = 2 + K         'To draw a line (7 blocks long) north
                           'east from endpoint of previous line
40    A(2) = 3 + K         ' ... etc.
50    A(3) = 4 + K
60    A(4) = 5 + K
70    A(5) = 6 + K
80    A(6) = 7 + K
90    A(7) = 8 + K
95    A(8) = 0             'Terminate drawing
100   CLS: DRAW SET @ 64,23 USING A
110   GOTO 110
```

The program sets up the numbers that specify the shape of an octogon in lines 10 through 95. Line 100 clears the screen and actually SETs an octogon on the screen, starting the first line at 64,23. Try typing and executing this program to get an idea of how DRAW works. Of course, DRAW RESET .... would CLEAR a figure from the screen.

Other graphics codes are available by adding an offset to the normal code of 1 through 8:

Codes	Offset	Effect
9 to 16	+8	Point is unconditionally SET, regardless of <flag> specified in DRAW
17 to 24	+16	Point is unconditionally RESET
25 to 32	+24	The current drawing point is updated without points being either SET or RESET

These additional codes just help to make DRAW more versatile.

Any figure can be rotated or expanded in scale, depending on the contents of two memory locations (on the Model I.) The locations are:

Address	Function
16426	ROTATION FACTOR (initially 0). This offset is added to each direction code, in effect "rotating" a figure by any multiple of 45 degrees
16427	SCALE FACTOR (initially 1). Each iteration byte in a graphics code is multiplied by the contents of this memory location

For example,

```
POKE 16427,2
```

expands the scale of any figure by two, making it appear twice as large. And,

```
POKE 16426,1
```

rotates any figure clockwise by 45 degrees. By altering these memory locations quickly and DRAWing, the effect of animation can be achieved.

Model III ENHBAS is slightly different from Model I ENHBAS. Since the Model I memory locations are used by the Model III's resident system routines, two additional commands were added: ROT and SCALE. Instead of POKE 16426,x merely use: ROT = x. Instead of POKE 16427,x merely use: SCALE = x.

PLOT (SET or RESET a line or box on the screen)

```
PLOT <flag>,   x1,y1   TO   x2,y2
     where <flag> denotes the kind of plot:
```

<flag>	action
S	SET line between two points
R	RESET line between two points
SB	SET BOX with points as opposite corners
RB	RESET BOX with points as opposite corners

and where x1,y1 and x2,y2 represent graphics points in normal SET/RESET format: X coordinate, Y coordinate

PLOT draws or erases a line or a box between any two points on the screen. PLOT can be very useful for drawing fast graphics.

EXAMPLE

```
10     CLS
20     PLOT S, 0,0 TO 127,47
30     PLOT S, 0,47 TO 127,0
40     PLOT SB, 0,0 TO 127,47
```

In the above example, line 20 draws a diagonal line from the upper left hand corner to the lower right corner; line 30 draws a diagonal line from the lower left corner to the upper right corner; and line 40 draws a box around the entire screen.

One useful option is replacing x1,y1 with an up-arrow, which designates that the PLOT should begin at the end of the last PLOT. For example:

```
10     PLOT S, 0,0 TO 127,0
20     PLOT S, <up-arrow> TO 127,47
```

In line 10, a line is drawn across the top of the screen. In line 20, a line is drawn from the last point PLOTed (127,0) to (127,47), giving a line down the right side of the screen.

INVERT (Invert all graphics on the screen)

```
INVERT
     no operand accepted
```

INVERT very simply takes all graphics characters (including blanks) and turns white to black and vice-versa. Non graphics characters, such as letters of the alphabet, are ignored.

EXAMPLE

```
10    DEFINT A-Z
20    PLOT S, 0,0 TO 127,47          ' Draw line, corner to corner
30    PLOT S, 0,47 TO 127,0          '   "    " , opposite corners
40    PLOT SB, 0,0 TO 127,47         ' Put box around whole screen
50      FOR T=1 TO 100: NEXT         ' Delay
60    INVERT                         ' Invert graphics
70    GOTO 50                        ' ... and loop
```

LEFT ("scroll" screen LEFT)

```
LEFT
     no operand accepted
```

LEFT simply moves the contents of the entire screen left. For instance, try typing:

```
FOR X=1 TO 63: LEFT: NEXT
```

with a lot of information on the screen. The effect is interesting. The main purpose of LEFT is to provide moving graphs on the screen. All you have to do is to PRINT@ information somewhere near the right edge of the screen, then do a LEFT as many times as necessary. This way, you could conceivably have, say, a profit curve scrolling to the left as each point was plotted.

PRINTER RELATED COMMANDS

CLM (C o L u M n Set printer width)

```
CLM = nmexp
     where nmexp is a numerical expression denoting
     maximum printer width, ranging from 7 to 255
```

CLM allows you to set the maximum printing width on your printer. For example:

```
CLM = 79
```

automatically rolls the printer over if a print line goes past 79 characters. CLM is not available on the Model III.

PAGE (Set page rollover length)

```
PAGE = nmexp
     where nmexp is a numerical expression denoting
     the maximum number of lines to print
```

PAGE allows you to set the maximum number of lines printed per page on your printer. For instance, to set the printer to print a maximum of 60 lines per printer page, simply execute: PAGE = 60. To disable this automatic top of form, simply set PAGE = 255. PAGE is not available on the Model III (disk users can invoke the Model III TRSDOS command FORMS.)

MISCELLANEOUS COMMANDS

WPOKE (16 bit POKE)

```
WPOKE exp1, exp2
     where exp1 is a memory address
       and exp2 is a 16 bit integer to place in the address
```

WPOKE is a 16 bit version of POKE. It essentially POKEs two values at once, a high byte and low byte, usually for two byte address changes in memory. Also see WPEEK under functions.

PLAY (play music over the cassette output jack)

IN VERSION 1.8 AND LATER VERSIONS

```
PLAY <var$>
     where <var$> is a string variable
```

With PLAY, you can play many melodies over a speaker/amplifier connected to the cassette output jack of your TRS-80.

The notes to be played are specified by sound codes contained in any specified string variable (NOT string expression, but any string variable.) Each note takes three bytes to specify: The tone byte, and the length bytes.

The tone byte can be between 0 and 255, with 1 specifying the "highest" note and 255 or 0 the "lowest" note, and all variations in between 1 and 255.

The length can be between 0 and 65535, making PLAY very versatile and capable of playing a great deal of songs.

Let's say the tone is represented by the variable "T" and the length by "L". Then:

```
1000  . . .
1010  LH = INT( L/256): LL = L - 256 * LH
1020  P$ = CHR$(T) + CHR$(LL) + CHR$(LH)
1030  PLAY P$
1040  . . .
```

In this example, a note would be played, specified by "T" and "L". You can string notes together to be played sequentially, merely by concatanating the note codes together. P$ might already hold 3 notes. So, in line 1020: P$ = P$ + ... would be valid, to string yet another note on. At line 1030, a 4 note melody would be played.

The easiest way to play a song, however, is to fill each element in a string array with one note, and use a FOR-NEXT loop to PLAY each individual note. By using integer variables, this should be quick enough for many songs.

SCROLL (Scroll protect lines on the screen)

```
SCROLL = SET
       Clears screen an turns scroll protection ON
       (Unnecessary with Model III ENHBAS)

SCROLL = RESET
       Turns scroll protection OFF
       (Unnecessary with Model III ENHBAS)

 SCROLL = <number of lines>
      where <number of lines> is a numerical expression
      denoting the number of lines to be scroll protected.
      Must be between (inclusive) 0 and 13 on the Model I
       or between 0 and 7 (inclusive) on the Model III
```

Many times you'll want to have a stationary heading at the top of the screen with information scrolling underneath it. SCROLL allows you to do this.

EXAMPLE

```
10    CLEAR 500: X=1
20    A$ = "THIS IS A TEST OF THE SCROLL PROTECT"
30    CLS
40    PRINT TAB(10) "THIS IS A HEADING TO BE PROTECTED"
50    PRINT TAB(14) "NOTICE HOW THE SCREEN SCROLLS OFF"
60    PRINT TAB(19) "UNDERNEATH THIS HEADING"
70    PRINT STRING$(63,45)
75    SCROLL = SET
80    SCROLL = 4:                   'Protects the top 4 lines
90     FOR I=1 to 25
100     PRINT TAB(9) A$; I
110      FOR T=1 to 75: NEXT T:     ' DELAY
120   NEXT I
125   SCROLL = RESET                ' DISABLE SCROLL
```

Note that lines 75 and 125 should be OMITTED on the Model III version of ENHBAS. In place of line 125, SCROLL = 0 is sufficient to disable the scroll protect.

OUTPUT (Send output to printer or screen depending on expression)

```
OUTPUT (exp) information

     where exp is any numeric expression
     If exp = 0, then OUTPUT is acting as a PRINT command
     If exp = 1, then OUTPUT is acting as an LPRINT command
```

OUTPUT simply allows you to use a variable to determine whether information is to be sent to the screen or to the printer. For example,

```
OUTPUT(A) "THIS IS A TEST" TAB(40) "OF OUTPUT"
```

is equal to

```
PRINT "THIS IS A TEST" TAB(40) "OF OUTPUT"
```

when A = 0, and is equal to LPRINT ... when A = 1. Any expression other than 0 or 1 will generate an FC ERROR.

ZSTEP (Turns program single stepping on and off)

```
ZSTEP = <flag>

     where <flag> is a numeric expression
     If <flag> = 0, single stepping is turned off
     If <flag> = 1, single stepping is enabled
```

This simply allows you to pause in between execution of every program statement. It is normally off. ZSTEP = 1 turns single stepping on. Thereafter, before an instruction is executed, there will be a pause. Hitting any key causes the instruction to be executed. The cycle repeats until ZSTEP = 0 turns single stepping off.

EXEC (Execute a string expression as a program statement)

```
EXEC (<exp$>)
    where <exp$> is a string expression, which will be
    executed as if it were an actual program line
```

EXEC is an extremely powerful and potentially useful command that is usually found only in the languages of large computers. It allows you to take any string expression and EXECute it as if it were a program line that you typed in. The potential is limitless.

Any legal program statement can be represented by a string expression and executed with EXEC. Any errors in the executed code will be shown as occuring in the line that the EXEC is in.

Important Note: Do NOT use another EXEC or EVAL (see functions) inside an EXEC!

EXAMPLE

```
10   CLS
20   A$ = "FOR X=0 TO 10"
30   B$ = ": PRINT X;: NEXT"
40   EXEC (A$+B$)
50   'Line 40 is the same as: FOR X=0 TO 10: PRINT X;: NEXT
```

As you can see, string variables can be concatenated together to produce a complete program statement. Note that ENHBAS as well as BASIC commands can be executed, with the exception of another EXEC or EVAL, as noted previously.

NEW FUNCTIONS

CONSTANTS

PI (Returns the value of PI)

```
PI
     no operand accepted
```

This constant returns the value of PI, limited by the precision of the expression being evaluated. For example,

```
A! = PI
```

assigns A! to 3.14159. However:

```
A# = PI
```

assigns A# to the double precision value of PI, 3.141592653589793.

EN (Returns the value of "e" (base of the natural log))

```
EN
     no operand accepted
```

This constant returns the value of "e", limited by the precision of the expression it's used in.

```
Single precision: 2.71828
Double precision: 2.718281828459045
```

STRING FUNCTIONS

BIN$ (Convert numeric expression to 16 digit binary string)

```
     BIN$(<number>)
          where <number> is an integer numerical expression
```

This function returns the binary equivalent of a decimal number, expressed as 16 digits in string form. For example,

```
          A$ = BIN$(10432)
```

will assign A$ to "0010100011000000".

HEX$ (convert integer expression to 4 digit hex string)

```
     HEX$(nmexp)
          where nmexp in an integer expression to be converted
          to the hexadecimal (base 16) equivalent
```

HEX$ is similar to BIN$, except that is converts an expression to base 16 instead of base 2. For example,

```
          PRINT HEX$(-2)
```

would print

```
          FFFE
```

on the screen. Of course, HEX$ can be used inside any legal string expression, such as A$+HEX$(B)+Q$.

WINKEY$ (Wait for keyboard input)

```
WINKEY$
     no operand accepted
```

WINKEY$ operates like INKEY$, except that it automatically loops until a key is hit, unlike INKEY$. For example,

```
10     A$ = WINKEY$
```

is functionally equivalent to:

```
10     A$ = INKEY$
20      IF A$ = "" THEN 10
```

MISCELLANEOUS FUNCTIONS

CALL (Call a machine language routine)

```
CALL <address> , <argument>
     where <address> is a numerical expression representing
        a memory location
     and <argument> is an integer numerical expression
        to pass in the HL register to the routine
```

Unlike USR calls, the CALL function allows you to call a machine language routine without predefining an address. Integer parameters may be passed both ways. For example, if you had a machine language routine at 9000 hex, with HL passing parameters and results to and from the routine, you would simply do:

```
X = CALL &H9000, A
```

where A is the parameter to be passed in HL. The BASIC variable X would be set equal to whatever the HL register held when a machine language RET was executed at the end of the machine routine.

CALL cannot be executed as a direct command.

EVAL (EVALuate a string expression as a numerical expression)

```
EVAL (<exp$>)
     where <exp$> is a string expression
```

This command can be very useful. It is an enhanced VAL function. It takes any string expression or string literal and evaluates it, and then treats it as an algebraic expression. For example,

```
10    X = 3
20    A$ = "X * X + 4"
30    A = EVAL (A$)
```

will set A equal to 13. It is equivalent to:

```
10    X = 3
20    A = X * X + 4
```

Now, it is possible to input algebraic expressions in string variables and EVALuate them, instead of forcing the user to type an equation into a program line. EVAL can be very useful for educational applications.

IMPORTANT: Do NOT use another EVAL inside of an EVAL.

SIZE (Returns the size of the resident ENHBAS program)

```
SIZE
     no operand accepted
```

SIZE returns the size of your current program, in bytes.

WPEEK (16-bit PEEK of memory)

```
WPEEK (<address>)
     where <address> is an integer numerical expression
     representing a memory location
```

WPEEK's primary function is to return a two byte memory address. WPEEK (X) is functionally equivalent to:

```
PEEK (X) + 256 * PEEK (X+1)
```

For information on placing 16 bit numbers in memory, see WPOKE.

APPENDIX A

New ERROR messages / codes

The additional commands of ENHBAS require new error codes, from 240-246. Error trapping routines (with ON ERROR GOTO) can trap these new errors easily because the system variable ERR will hold the error code, as with illegal function calls, etc.

ERR CODE	ERROR MESSAGE	MEANING
240	TAG STACK FULL	Attempt made to TAG more than 14 arrays at once
241	NO SORT KEYS GIVEN	Attempt to SORT without previously KEYing arrays
242	UNDEFINED LABEL	Attempt to jump to an undefined line label
243	PRINTER NOT READY	The line printer was not ready
244	KEY STACK FULL	Attempt made to KEY more than 14 arrays at once
245	WEND WITHOUT WHILE	WEND executed without previously executing WHILE
246	ATOP GREATER THAN SORT KEY DIMENSION	An ATOP expression was equal to or greater than the dimension of of the KEYed array(s)

INDEX

NEW RESERVED WORDS

SPECIAL Tape enhancements to CASSETTE ENHBAS (Model I/III)

The cassette version of ENHBAS provides you with some of the normal disk BASIC features as well as the new ENHBAS instructions and utilities:

1) MID$ (var$, n1 <,n2>) = exp$

where	var$	is a string variable to be modified
	n1	is the starting position to change
	n2	optionally specifies how many characters to replace in the string variable
and	exp$	is any string expression

Instead of the function MID$ which returns a substring of a string variable, the command MID$ places a string expression into a specified string variable, replacing characters. If A$="ABCDE", then after the execution of MID$(A$,2,3)="XYZ", A$ would be "AXYZE".

2) INSTR(<n,> exp1$, exp2$)

where n	optionally specifies the place where the search for exp2$ is to begin in exp1$
exp1$	is a string to be searched
exp2$	is a string to search for in exp1$

INSTR is a function which searches a specified string expression for an internal substring. For instance, if A$="THIS IS A TEST", and B$="IS", then PRINT INSTR(A$,B$) would start searching at position 1 in A$ for B$, and finally find it at position 3. Thus, a 3 would be printed. Optionally, the initial search position can be specified. So, PRINT INSTR(4,A$,B$) would print a 5.

3) LINEINPUT <"prompt";> var$

where "prompt"	is an optional prompting message
and var$	is any string variable

LINEINPUT inputs only to string variables. It does NOT check for delimiters such as "," or ";". Therefore, any character may be input without fear of an ?EXTRA IGNORED message.

4) &Hdddd, &Odddddd, &Bd0-d16

where "d" is a digit

These functions allow you to use hex, octal, or binary constants in numerical expressions. For instance,

```
PRINT &H67ED
```

would print a 26605. Octal constants and binary constants may also be used.

Also, some of the short-entry commands must be changed from page 10:

0	-	GOTO	5	-	CSAVE"
1	-	EDIT. <ENTER>	6	-	CONT <ENTER>
2	-	RUN <ENTER>	7	-	LINEINPUT
3	-	GOTO 10 <ENTER>	8	-	PLOT
4	-	CLOAD	9	-	DRAW