VZ200/300 - Extended Basic "XB2" by Russell Harrison

Contents

General Information

To load Extened Basic, make sure the Extended Basic disk is in the disk drive and type:
BRUN"XB"
The drive will spin for about 15 seconds, a messgae should appear and the READY prompt and cursor will come back.
Extended Basic commands are used in exactly the same way as normal commands. However, lines containing Extended Basic commands should only be typed in after Extended Basic is loade. Extended Basic can be loaded at any time, and will not destroy a Basic program that is already in memory.
Using expressions in GOTO statements, disk commands etc.
Extended Baisc lets you write things as:

```
GOTO A*10
LOAD N$
```

This will work for GOTO, GOSUB and IF..THEN..ELSE statements, and all disk commands. In the RUN"<filename>" command, the expression must stat with a double-quote, for example:

```
RUN"FILE"+STR$(N)     is permitted
RUN N$                is not permitted but
RUN""+N$        is PERMITTED.
```

Similiary, expressions in IF..THEN..ELE statements must start with a number.
Note: Throughout this manual, angle brackets -<and>- have been used to show parts of a statement that should not be typed directly, but replaced with a number, variable, or whatever.

VARIABLES

Double Precision Arithmetic
Extended Basic allows the use of 15 diget "double precision" numbers. Any normal numbers with more than six significant digits will be treated as double-precision. A # symbol after the number can also be used to indicate

double-precision. In double precision scientific-notation, a "D" should be used instead of the "E" used to indicate the exponent.

Example

```
1.513          is single-precision
1.513236578          is double-precision
1.92567-10     is single-precision
1.925674523d-10 is double-precision
35#            is double-precision
```

Double-precision arithmetic will only be used if at least one of the operands is in double precision, for example:

```
PRINT 1/3
0.333333
PRINT 1#/3
0.333333333333333
```

Warning: SIN, COS, LOG, and most other functions only have 5-6 digit precision, even with double-precision numbers.

Double-precision variables

Even if an epression is worked out in double-precision, it will be reduced to single-precision if you attempt to store it in a normal variable. To solve this, double-precision variables are also allowed. A double-precision variable is indicated by a # after its name, eg:

```
A#
DH#
GAINS#
```

Double precision variable are used in the same way as other variables except they can not be used in FOR statements.

CINT((arg)) / CSNG((arg)) / CDBL((arg))

These functions convert a number to either integer, single precision or double-precision format, respectively, for example:

```
A=6:b=1
PRINT A/B
0.166667
PRINT CINT(A)/B
0.166666666666667
```

Remembe that double-precision arithmetic is only used if one operand is already in double-precision, so

```
PRINT CDBL(A/B)
```

Will not return the same value as the second example above.

DEFINT<var> / DEFSNG<var> / DEFDBL <var> / DEFSTR <var>

It can become tedious typing in %,$ and # signs for integer, string and double precision variables. The DEFINT, DEFSNG, DEFDBL, and DEFSTR commands let you permantly define certain variables as integer string or whatever. Variables are redefined according to which letter they start with. For instance,

```
DEFINT I
```

Will define as integer the variable I, and also any two letter variables starting with I, such as I5, Ia, Ik, etc. More than one letter can be defined in one statement, and ranges of letters can be specified by using a dash to represent ‘to’.

For example:

```
DEFINT I-L      defines as integer any variable starting with I,J,K or L.
DEFDBL A,X,Y    defines as double precision variables bginning with A,X, or
Y.
DEFSTR H        Defines as string variables starting with H.
H="HELLO"       (the $ sign is no longer needed)
```

DEFINT, DEFSNG, DEFDBL, DEFSTR are overridden by a symbol such as $ or #. To use the single precision form of a variable that has been defined as something else, an exclamation mark (!) should be used, for example:

```
DEFINT A
A!=1.5
```

Basic treats A%, A!, A#, and A$ (for example) as being completely different variables. The variable A (without a type sign) will refer to one of these, depending on the current type set by DEFINT, etc

This means that its value could change, for example:

```
A=1:A#=1.234
DEFDBL A:PRINT A
1.234
```

PROGRAMMING & DEBUGGING AIDS

AUTO <starting line number>,<increment>

When typing in a program, line numbers usually go up in steps of 10, 100 etc, and typing in each one wastes time. AUTO is a commmand which prints line numbers automatically, leaving the cursor on the same line for you to type something in. If a line already exists, it will be displayed as well as the line number. Either the line increment or both parameters may be omitted. The default for both is 10.

Example:

```
AUTO            Begins numbers at 10, increasing by 10
AUTO 100        Begins at 100, increasing by 10
AUTO 90, 20           Begins at 90, increasing by 20s.
```

DELETE<starting line>-<ending line>

This command deletes a block of lines from a program.

Example:

```
DELETE 1000-1090Removes statements between lines 1000 and 1090 (inclusive).
```

MOVE TO <line number>,<start>-<end>

This command copies a block of lines from one line number range to another. The lines being moved will overwrite any existing lines with the same line nmbers.

Example:

```
MOVE TO 1000, 60-700
```

RENUM <starting line number>,<increment>

Often, after modifying a basic program, you are left with a messy arrangement of line numbers which, besides looking bad, makes further modification difficult. RENUM can be used to renumber the program, back to a more orderly arrangement. Line numbers in GOTO, GOSUB etc statements will also be adjusted.

In the same way as with AUTO, one or both parameters may be left out.

For example:

```
RENUM      Results in line nmbers 10,20,30…
RENUM 100       Results in line numbers 100,110,120…
RENUM 100,40    Results in line numbers 100,140,180…
```

A ‘$’ sign within the program can be used to mark line numbers which should not be changed. (important subroutines etc). It should come between the line number and the rest of the line. The ‘$’ sign will be ignored while the program is running.

Example:
(before)
```
1 CLS
5 PRINT "HELLO"
6 GOSUB 1000
10 END
1000$ A=100:S=75:G=2
1005 N$="ABCDEFGHJKLMNOPQRSTUVWXYZ"
1010 PRINT N$;N$;N$;N$
1020 RETURN
```
(after RENUM)
```
10 CLS
20 PRINT "HELLO"
30 GOSUB 1000
40 END
1000$ A=100:S=75:G=2
1010 N$="ABCDEFGHJKLMNOPQRSTUVWXYZ"
1020 PRINT N$;N$;N$;N$
1030 RETURN
```

FIND"<string>"
This command searches thorugh a program unitl it finds a line which contains the string required. This line will then be displayed, and the computer will wait for you to press [SPACE]. It will then do the same for the next occurrence, and so on.
Example:
```
FIND "IF A=1 THEN"
130 IF A=1 THEN PRINT "YES"           (press [SPACE])
410 IF A=1 THEN J=J+1           (press [SPACE])
READY
```
FIND "<string>",<start line>-<end line>
FIND can be told to search only a particular range of line numbers
Examples:
```
FIND "BYE",1000-2000  Searches lines 1000 to 2000
FIND "BYE",500-       Searches lines from 500 onwards
FIND "BYE",-3000      Searches up to line 3000
```

TRON and TROFF
To aid debugging, Extended Basic can display a "trace" – the line numbers of each statement as they are executed. This feature is turned on and off by the TRON and TROFF commands.
Example:
```
TRON
READY
RUN
<10><20><30><40><20>…..
READY
```
The SPEED command (see "Other commands") may be used to slow down the screen output.

COMPRESS
This command removes all spaces and comments (in REM statements) from a program, to save memory and speed up execution. The "stub" of a REM statement

is left, in case it is referred to by a GOTO command.
Example:

```
10 REM RANDOM BEEPS
20 BEEP RAND(255,5,RND(2))
30 GOTO 20
COMPRESS
READY
LIST
10 REM
20 BEEP RAND(255,5,RND(2))
30 GOTO 20
```

OLD
This command reverses the action of a new statement. It can only be used if a new program has not been loaded or typed in – if OLD is typed with a program still in memory, it will be left as it is.
Example:

```
10 PRINT "VZ EXTENDED BASIC"
20 GOTO 10
NEW
LIST
READY
OLD
READY
LIST
10 PRINT "VZ EXTENDED BASIC"
20 GOTO 10
```

GRAPHICS

PLOT (<x1>,<y1>) TO (<x2>,<y2>)…
This command line draws a line, in MODE(1), between two or more points.
For example:

```
PLOT (0,0)TO(127,63)  Draws a diagonal line across the screen.
PLOT (10,10) TO (10,54) TO (118,54) TO (118,10) TO (10,10) Draws a rectangle
```

FILL (<x>,<y>)
This command fills enclosed space on the screen, starting from a particular point.
For example:

```
10 MODE(1)
20 PLOT (0,0) TO (127,0) TO (0,63) TO (0,0)
30 FILL (5,5) : REM FILL IN TRIANGLE
40 GOTO 40
```

FILL (<x>,<y>) TO <colour1>,<colour2>….
This version of FILL allows you to specify which colours form the border, and which colours to cover over.
Example:

```
FILL (5,5) TO 3       Only stops at blue lines
FILL (5,5) TO 3,4           Stops at blue or red lines
```

DRAW <str array>,(<x>,<y>)
This command allows you to define graphic shapes, and display them faster and

more easily than with SET. The shape should be held in a one-dimentional string array, with each individual string representing one line on the display. Each character in each string represents on epixel. The characters which may be used, and their meanings are listed below:

1,2,3,4	- colours green, yellow, blue, red
#	- current colour set by COLOR command
[SPACE]	"transplarent" (leaves the colour of that pixel the same

Example:
```
100 REM ROCKET
110 DIM S$(8)
120 S#(0)=" #"
130 S#(1)=" ###"
140 S#(2)=" 333"
150 S#(3)=" ###"
160 S#(4)=" ###"
170 S#(5)=" #####"
180 S#(6)=" 33333"
190 S#(7)=" #####"
200 S#(8)=" 44 44"
210 MODE(1)
220 FOR C=2 TO 4
230 COLOR C
240 DRAW S$,(C*10,52)
250 NEXT
260 GOTO 260
```

ERASE <str array>,(<x>,<y>)
This command is used to remove a shape, drawn by DRAW, from the screen. Whenever DRAW would have plotted a point, ERASE will reset it to the background color.
For examplw, adding this line to the program above will remove the middle rocket:
```
255 ERASE S$,(30,52)
```

XDRAW <str array>,(<x>,<y>)
This command is similar to DRAW, except that it does not overwrite the background, but alters its colours so that

a. If no colours other than background colour are present beforehand, the resulting picture will be the same as with DRAW, and
b. If the same shape is XRAWn in the sample place again, the original background will be left as it is was before the first XDRAW.

Example
```
100 DIM D$(4)
110 D$(0)=" #"
120 D$(1)=" # #"
130 D$(2)="# #"
140 D$(3)=" # #"
150 D$(4)=" #"
160 MODE(1):COLOR 3
170 FOR I=1 TO 4
180 FOR X=0 TO 120 STEP 6
190 XDRAW D$, (X,0)
200 NEXT:NEXT
```

```
210 GOTO 210
```

CIRCLE (<x>,<y>),<diameter>
This command draws a circle with its center at (<x>,<y>). The program compensates for the VZ’s rectangular pixels. The diameter should be a number of pixels from top to bottom, not from side to side.
Example:

```
CIRCLE (63,31),20
```

BACKGR <color>
This command is used to change the background color of the sceen (in MODE(1)). BACKGR would normally be used immediately after a MODE(1) command, but if a picture already exists, then its shape will be preserved but its colors modified in the same way as with XDRAW.
For example

```
10 MODE(1)
20 PLOT(10,10) TO (120,60) TO (10,10)
30 FILL (15,30)
40 BACKGR RND(4)
50 FOR I = 0 TO 300:NEXT
60 GOTO 10
```

SOUND

BEEP <pitch>,<duration>
This command is a more flexible version of Basics SOUND command. <Pitch> and <duration> may be any number between 1 and 255.
Example:

```
BEEP 255,10          High pitched, short note
BEEP 50,255          Long, fairly low pitch
```

BEEP <pitch1> TO <pitch2>,<duration>
This version of BEEP command varies the note to give a "sliding effect"
Example

```
BEEP 100 TO 200,255 "slides" the pitch of the note from 100 to 200
```

BEEP <Pitch>,<duration>,<volume>
A third parameter may be included in a BEEP statement – volume. This may be either 1 or 2 (the default is 2)
Examples

```
BEEP 40,20,1
BEEP 10 TO 200,255,1
```

NOISE <intensity>,<duration>,<volume>
This command is used to produce a "rushing" or scraping sound, depending on the intensity. It has the same format as BEEP command.
Example:

```
NOISE 100,20
NOISE 150,40,1
NOISE 1 TO 250,70
NOISE 1 TO 250,70,1
```

FORMATTED INPUT

INPUT ! ….
Putting an exclamation mark (!) in an INPUT statement will stop the computer from printing the ? propmpt.
For example:

```
10 INPUT ! "ENTER FILE NAME: ";F$
RUN
ENTER FILE NAME:
```

INPUT @ <position>; …
Extended Basic lets you use the @ command (as in PRINT @) in an INPUT statement.
Example: INPUT @64;"NAME";N$
INPUT USING <characters>; …
This command lets you control which characters can be typed in an INPUT statement. These characters may be specified in three ways:

- as a string eg. INPUT USING "ghjk" (more than one character can be included in the string)
- as an ASCII code, eg. INPUT USING 65,66,67
- as a range of characters or codes, eg. INPUT USING "a" to "z"

These three formats may be present in the same statement, eg
INPUT USING 8.9.10.13."0 TO 9)","ABCDFEG"; … Note that control codes (backspace, etc – this includes [RETURN] are not automatically included.
ASCII codes for these are as follows:

Key	Code
Backspace	8
Forward space	9
Up-arrow	27
Down arrow	10
Insert	21
Rubout	127
Return	13

This statement will include all of them:
INPUT USING 1 TO 27, 127, ….
USING, ! and @ commands may all be included on the same line. Note that they may be in any order, but a prompt must always come at the end line.
Example:

```
INPUT ! @0; USING 1-31,127,"YN"; "CONTINUE (Y/N) ";Q$
```

STR INPUT <str variable>
A normal INPUT statement scans through the line typed in and breaks it up according to the commas in it. This allows more than one item per line, but it is annoying in that a comma cannot be treated like other characters. STR INPUT is a version of the INPUT statement which only ever accepts one string, but ignores commas. It can be used with !,@, and USING as well as with a prompt.
Example:

```
10 STR INPUT "CATEGORIES REQUIRED";C$
```

GET <string variable>
This command accepts one character from the keyboard. The cursor flashes

while the computer is waiting, and the usual key-beep will sound as it is typed. GET can be used with USING, @, etc
Example:
```
10 GET USING "S"; "HIT <S> TO START";Q$
```

USER DEFINED FUNCTIONS

Exteneded Basic allows you to create your own functions. (A function is something which when given one or more values, returns an answer SIN, LOG, etc are functions). The name of each function mus begin with FN, followed by a latter and then any number of alphanumeric characters. Only the first two characters are significant.
DEF
The DEF statement is used to define a function. DEF statements must be in program, not typed directly.
Example:
```
10 DEF FNC(F) = (F+32)*5/9 : REM FARENHEIT TO CELCIUS
20 DEF FNAVM5(A,B,C,D,E)=(A+B+C+D+E)/5
```
Functions are used just like SIN, COS, etc.
For example:
```
PRINT FNC(100)
37.7778
```
The variables used to hold a functions parameters normally do not affect other variables of the same name, if an error occurs while the function is being evaluated, however, the tempory value will be left in each variable.
For example:
```
F=1.57
PRINT FNC(100)
37.7778
PRINT F
1.57
```
A function may return a string instead of a number. Parameters may also be integers, strings, or double precision.
Example:
```
10 DEF FNRJ(T,N$) = RIGHT$(REP$(T<" ")+N$T)
20 PRINT FNRJ(10,"HELLO")
RUN
HELLO READY
```
External variables can also be used in a function.
For example:
```
10 PI = 3.1415926536
20 DEF FNAREA(R#)=PI*R#*R#
```

Finally, a function may have any number of parameters, although it does not necessarily have to have any.
For example:
```
10 DEF FNMEM = FREE + SFREE
20 DEF FNDIST(X1,Y1,X2,Y2) = SQR((X2-X1)^2+(Y2-Y1)^2)
```

STRING HANDLING

INSTR (<string1>,<string2>)
This function searches through <string 1> for the first occurrence of <string 2>. If it is found, it’s position in <string 1> will be output, otherwise zero will be the result.
Example:

```
PRINT INSTR("HELLO THERE","TH")
7
PRINT INSTR("HELLO THERE","IN")
0
```

Using MID$ on the left hand side of an equation:
A small part of a string variable can be changed by using a MID$ command, on the left side of an assignment statement. The length of the sub string is determined by the right hand side of the equation, and a third parameter (the length in normal MID$ function) does not need to be included. (If it is it will be ignored)
Example:

```
MID$(A$,%)="FROG"
MID$(T$,2,3)="QUE"
```

ERROR TRAPPING

Normally errors in a program generate an error message and stop the program from going further. Extended Basic allows you to write your own error handling subroutines, which does not necessarily have to stop the program.
ON ERR GOTO <line number>
This is the statement which tells the computer where your error handling routine is. To disable that subroutine use ONERR GOTO without a line number. More than one ONERR GOTO may be used if different subroutines must be used for differenet sections.
RESUME
or
RESUME <line number>
or
RESUME NEXT
This statement should be placed at the end of your error handling subroutine to transfer execution back to the main program. RESUME returns to the statement that generated the error; RESUME NEXT goes back to the statement after the error, RESUME <line number> goes back to the line you specify. If a RESUME is not included a "NO RESUME ERROR" will result.
Example

```
10 ONERR GOTO 1000
20 INPUT "PITCH";P\
30 BEEP P,30 : REM GENERATES AN ERROR IF P>255 OR P<1
40 ONERR GOTO
1000 PRINT "THAT NUMBER IS OUT OF RANGE".
1010 PRINT "PLEASE ENTER ANOTHER ONE"
1020 RESUME 20
```

ERL and ERR
When an error occurs, the computer stores the line number and the type of

error in two variables, ERL and ERR. These can be used in your error subroutine. ERL returns the number of the line with the error in it, while ERR returns a code representing the type of error. These codes are as follows:

CODE	Error Message
0.	Next without for error
2	Syntax Error
4	Return without gosub error
6	Out of data error
8	Function code error
10	Overflow error
12	Out of memory error
14	Undef'd statement error
16	Dad subscript error
18	Redim'd array error
20	Division by zero error
22	Illegal direct error
24	Type missmatch error
26	Out of space error
28	String to long error
30	Formula too complex error
32	Can't cont error
34	No resume error
36	Resume without error
38	-
40	Missing operand error
42	Bad file data error
44	Illegal command error
46	File already exists
48	Directory Full
50	Disk write protected
52	File not open
54	Disk I/O error
56	Disk fukk
58	File already open
60	-
62	-
64	Unsupported device
66	File type mismatch
68	File not found
70	Disk buffer full
72	Illegal read
74	Illegal write

DISK COMANDS

UNERA "<type>","<filename>"
This command allows you to bring back accidently errased files. If a file has been overwritten since it was erased, a "FILE
NOT FOUND" message will appear. The file type (T,B, or D) of the file must be included, since this is always removed by the
ERA command.

Examples:
```
UNERA "T", "PICTURE"
UNERA "D", "CODE"
```

The names of the files which have been errased can be obtained useing the XDIR % command. (see below)

XDIR

This command does the same as DOS's DIR command, except that files on the disk are listed in three columns, so more will fit on one screen. If more than one screenful exists, the computer will wait for you to press [SPACE] before continuing.

Example XDIR
```
T:FISH     B:ROCKET        B:TEST
T:FIRST    T:ANlMALS       T:SHIP
D:WORDS    D:WORDS2        T:PRINT
B:MOUSE
```

LDIR

This command is similar to XDIR, except that the listing will be printed on the printer, and in seven column is about the'width of an 80 column printer.)

XDIR"<filename" / LDIR"<filename"

These two commands will list particular groups of files, whose names match the file you typed in. Two "wild cards" are permitted.

[SHIFT]-[Z] (black rectangle) this character is revelant

[SHIFT]-[J] (coloured rectangle) the rest of the filename is irrevelent.

Example XDIR "FIREZ"
```
T:FIRE11 T:FIRE12 T:FIRE13
B:FIRE G
READY
XDIR "FOGJ"
T:FOG      T:FOGLIGHT      T:FOG6.19
READY
```

XDIR% / LDIR %

These two commands list to screen or printer, files that have been erased, but whose names still exist. The file types will appear as question marks(?). The files listed will not necessarily UNERAse succesfully, but a file which is not listed will always generate a "FILE NOT FOUND" message if this is attempted.

Example
```
XDIR%
?:HELLO    ?:NEXT     ?ABCDE
?:TEMP     ?:USELESS
```

ERALL"<filename"

This command deletes groups of files from a disk. The method of selecting files is the same as with XDIR"<filename>", and this fact can be used to check that the files YOu are about to erase are the ones you think they are.

ERALL "AB<SHIFT>Z". ERASES ALL FILES WHICH BEGIN WITH [AB]

MERGE"<filename>
This command combines two programes, one in memory and one on disk, into a single program, Lines will be automatically be arranged in the right order, and where the same line number exists in both original programs, the line originally held in memory willbe deleted.

LENGTH"<filename>"
This file function returns the length (in bytes) of any Basic or binary file.
Example:

```
PRINT LENGTH ("picture")
2903
PRINT LENGTH("answers")
152
```

TYPE$("<filename>")
This function returns the file type of the file<filename>.
For example:

```
PRINT TYPE$("CIRCUS")
B FILE$(«arg»)
```

The directory of a disk consists of 120 entries, each of which can contain a file name and the pointers to where it is
stored. FILE$ lets a program access these entries.
The value in brackets should be a number between O and 119 (inclusive) representing one directory entry. If a file name is stored in that cell, it will be returned, padded out to eight characters with spaces. If the entry contains the name of a file which has been deleted, then a string consisting of eight spaces will be returned. FinallY, if no file is stored there (that is, if the cell is past the last entry used), a null string ("") will be the result.

NEWDISK
To speed up access, LENGTH, TYPE$ and DILE$ load eight directory entries at once, and store these so the next seven entries
can be used without disk access. However, this also means a new disk will not necessarily be recognised. To avoid this, a NEWDISK statement should be placed before any use of these three functions, after a possible change of disks.
Note: This is only necessary in programs, not after LENGTH, FILE$, or TYPE$ statements typed directly from the keyboard.
NEWDISK, LENGTH, TYPE$. and FILE$ can be used to create customised directory listings. menues. ect.
For example:

```
10 PRINT "TYPE; TAB(6); "NAME"; TAB(17);"length":PRINT
20 NEWDISK
30 F=0
40 D$=FILE(F)
50 F=F+1 60 IFD$=22THEN END
70 IF D$" "THEN100
80 PRINT TYPE$(D);TAB(6);D$; TAB(;17);
90 IF TYPE(D$)<>"D"THENPRINT LENGTH(D$) ELSE PRINT "="
100 IF F<120THEN 40
```

DECIMAL HEXADECIMAL BINARY CONVERSION

&H<hex number>
Hexadecimal numbers of up to four hex digits can be used in Extended Basic by placing &H in front of them.
example:

```
PRINT &H1A
26
PRINT &H7000+&H15F
29023
```

Hexadecimal numbers greater than &h7FFF will come out negative so they can be used directly in PEEK statements. ect.

&B<binary number>
Binary numbers of UP to 16 bits can also be used, by using &B instead of &H.
EXAMPLE

```
PRINT &B10001011
139
```

HEX$(<arq>)
This function converts numbers into four digit hexadecimal format. If, for example only two digits are required then they can
be extracted using the RIGHT$ function.
Example:

```
PRINT HEX$(31465)
7AE9
A$=HEX$(F+G)
PRINT RIGHT$(HEX$ (255),2)
FF
```

BIN$(<arg>)
BIN$ is the same as the HEX$ function, except that it outputs a sixteen digit binary number.
Example:

```
PRINT BIN$(28674)
0111000000000010
```

HDEC(<hex string>) / BDEC(<binary string>)
These two functios convert hexadecimal and binary strings to decimal.
Example:

```
PRINT BDEC("10101010")
170
A$="6800"
PRINT HDEC(A$)
26624
```

MACHINE LANGUAGE SUPPORT

WPEEK(<address>)
This function is similiar to peek, except that it fetches two bytes of memory at a time, to form one two bit integer.
Example:

```
PRINT WPEEK(&H7820)
29152
```

WPOKE<address>,<data>
This command is the equivalent of POKE. except that it handles two bytes at a time.
Example:

```
WPOKE &H7820, &H7000
```

CALL <address>
This command is used for calling machine code subroutines from Basic.
Example:

```
CALL &H3450: REM USE ROM KEY BEEP ROUTINE
```

DEF USR<name>=<address>
This command can be used to set up several different machine code functions, which can be called at any time. The name of the subroutime may have any number of letters, but only the first two will count. The first character in the name must be a letter, and the other characters should be either alphabetic or numeric.
Example:

```
DEF USRA = &H1547: REM 'SIN FUNCTION
```

Functions are called like this:

```
PRINT USRA(1,2)
0.93203
```

N=USRA(2)
Standard USR functions can still be used in Extended Basic, but the address pointer in 30862-3 will be changed if an Extended
Basic USR call is used.

VAPTR(<var>)
This function returns the address where a particular variable is stored, for example:

```
PRINT VARPTR(Y%)
31470
```

Variables are stored in a variety of ways, depending on their type. Strings are stored as a length byte, followed by a two
byte pointer to the address of the actual string. If the string is the product of a string opperation, it will be stored in the string area near the top of memory. If it is a piece of 'raw' piece of data in a program, the pointer will refer to its location in the program. Strings can be used to

hold short, relocatable machine code subroutines, for example:

```
10 A$="abcdefghi" +"":REM DON'T WANT A MESSY LISTING
20 DEF FNAD=WPEEK(VARPTR(A$)+1)
30 FOR I=0TO8
40 READ H$:P=HDEC(H$)
50 POKE FNAD+I,P
60 NEXT 70 CALL FNAD
80 DATA 21,80,00,01,40,02,C3,5C,34
```

Warning: Basic may move a string during execution of a program, so it is best to use VARPTR every time you access a string,
or use a FN statement as I have here.
INteger variables are stored in two byte, signed integer format. (lsb-msb order)
Single precision variables are stored in the following formats:

```
POSITIVE NUMBERS:
BYTE  1.......   2.......   3.......   4.......
bit   0......7   0......7   0......7   0......7
NEGATIVE NUMBERS:
BYTE  1.......   2.......   3.......   4.......
<-lsb---mantissa msb->0 exponent
bit   0......7   0......7   0......7   0......7
<-lsb---mantissa msb->1 exponent
ZERO:
BYTE  1.......   2.......   3.......   4.......
bit   0......7   0......7   0......7   0......7
<----not used------------> 00000000
```

Note that the exponent is in base two, and has had an offset of 8OH added to it.
Double precision numbers are stored in the same way as single precision numbers, but with 7 bites mantissa instead of three.

OTHER COMMANDS

RANDOMISE
The random nuber generator in your computer will always generate the same sequence of numbers after you switch on the computer. Placing RANDOMISE at the start of a program will ensure that the same sequence of numbers will not occur whenever the program is run.

ON <exp> GOTO <line1>,<line2>,<line3>,<line4>,.....
Many programs include blocks of IF THEN statements like this:

```
100 IF A=1 THEN 1200
110 IF A=2 THEN 1350
120 IF A=3 THEN 1600
```

In Extended Basic that can be shortened to a statement like this:

```
100 ONA GOTO 1200,1350,1600
```

Any number of line numbers may be included. These will be used with the values 1,2,3,4,5,6,..... If there is no branch for a
number, the ON..GOTO statement will be ignored.
A GOSUB also may be used instead of the GOTO.

Examples:
```
ON X GOTO 100,200,300,400,500,600
ON F+G+1 GOSUB 90,130,170 ON INSTR("ECPLSVQ",C$) GOTO
1000,2000,3000,4000,5000,6000...
```

REP$(<number>,<character>)
This function can be used to produce long strings containing one character repeated over and over. The character can either be supplied as a one byte string or an ASCII code.
For example:
```
PRINT REP$(20,".")
....................
PRINT REP$(10,64)
@@@@@@@@@@
```

FREE
This function returns the amount of unused memory you have left.
Example:
```
PRINT FREE
9431
```

SFREE
This function tells you the amount of string space that is not used.
Example:
```
A$="HELLO"
PRINT SFREE
45
```

FIX(<arg>)
This function truncates the fraction part of a number. (INT does much the same thing, except that it always returns the largest integer less than the number, even on negative numbers.)

SPEED<speed>
This command allows you to slow down the rate that items are printed on the screen. The value of <speed> should range from 0
(normal display speed) to 255 (slowest).
Example:
```
10 SPEED 150: REM SLOW DOWN PRINTING RATE
20 PRINT"O..1..2..3..4..5..6..7"
30 SPEED 0: REM RETURN TO NORMAL
```

CSCR <character>
This command fills the screen with a particular character.
Examples:
CSCR "<shift>J" turns the screen black
COLOR 4: CSCR "<shift>Z" Turns the screen red
CSCR "A" covers the screen with As

RESTORE <line>
This version of Basic's RESTORE command lets you specify the starting line of a block of data.
For example:

```
10 DATA 1,5,9,19
20 DATA "HELLO","BYE","SORRY"
30 RESTORE 20
40 READ A$,B$: PRINT A$,B$
RUN
HELLO BYE
READY
```

CUSTOMISING EXTENDED BASIC

Extended Basic is made up of segments. To save memory, one or more segments may be left out when Extended Basic is loaded.
This is done using the ONLY and NOT statements for example:

```
BRUN"XB": NOT"DEFINE","SOUND"
BRUN"XB": ONLY"HIRES"
```

These commands will only work when Extended basic is being loaded. To save space, only the first two characters of each name
need be typed, for example:

```
BRUN"XB":NOT"HI","HE","DI"
```

The name length and function of each segmant is listed below.

NAME	BYTES USED	FUNCTION
Message	O	Prints opening message on the screen.
TEXT	497	Provides GET, STR, INPUT, INPUT USEING, CSCR statements.
HIRES	936	Provides PLOT, FILL, DRAW, X DRAW, ERASE. BACKGR, CIRCLE.
MC	43	Provides CALL, WPOKE, WPEEK.
HELP	1193	Provides MOVE, RENUM, FIND, MERGE, OLD, COMPRESS.
DEFINE	541	Provides DEF, FN, DEF USR.
CONVERT	193	Provides &H, &B, HEX$, BIN$, HDEC, BDEC.
DISK	1271	Provides XDIR, LDIR, UNERA, ERALL, NEWDISK, LENGTH. TYPE$, FILE$, and the use of string expressions as filenames
STRING	221	Provides INST, MID$, (on LHS of statement.)
SOUND	237	Provides BEEP, NOISE.

First of all BRUN "wp" and go to the disk directory and load "eb". After printing out "eb" load "eb2" and print that. That will be all the instructions except the index. I have printed the index in basic to be able to control the horizontal tabs. You may have to alter the printer controls to suit your printer. If your printer is EPSON compatiable it should print out OK. The LPRINTS after the horizontal tab settings are necessary on my printer. You may not require them. I have not included a front cover.
Jack Shearsmith.