Editing Keywords

ASSEM	Switches into assembler mode.
BANK *[bank[- bank][, on/off]]*	Display the current bank number, or change to the bank specified. If on/off specified will enable or disable the banks.
BASIC	Switches out of assembler mode.
COMP *["filename"[, devnum]]*	Will compile the program in memory and optionally save it to a file. *filename* must be 12 characters or less if specified.
DELETE *[start - end]*	Will delete a range of lines from *start* to *end* from your program. With no parameters, acts like NEW.
DESC *line#, label*	Create a subroutine *label* and start the code at *line#*.
ERROR	Will display the errors.
EXEC (>) *command block*	Will run a single line in immediate mode. Shorthand >
FAST	Enables speed up of some commands
FIND *text*	Searches the program in the current bank for lines containing *text*. Do not use quotes if searching for keywords. If searching for assembly, put a left bracket in front of the instruction.
LIST *[line#[- line#]]*	Will display the program in memory. Can optionally only display lines between the given parameters.
LISTER *[line#]*	A scrollable LIST. If *line#* specified, will start at that line.
LITE *[0/1]*	With no parameter or a 1, will enable LITE mode. 0 will disable.

Editing Keywords (cont)

LLIST *[line#[- line#[, printer?]]]*	Displays extended details about the program in memory. if the value of *printer?* is 1, the output will be sent to a printer.
NEW *[bank[- bank]]*	Clears the current program bank, or banks specified.
OLD *[bank[- bank]]*	Attempts to restore the program in the current bank, or banks specified.
PLIST *[line#[- line#]]*	Sends the program LIST to a printer.
QUIT	Quit Vision BASIC.
RENUM *start - end, new[, step]*	Will renumber the lines from *start* to *end* to *new*, using a *step* of 10 if not specified.
RUN *[line#]*	Runs the compiled, in memory program. If the program is not compiled or has been altered, will compile first. If *line#* specified, will start at that line number, otherwise will start from the first line of the program.
SLOW	Run at normal C64 speeds.
VLIST *[num]*	Will display all of the variables from the program in memory. If *num* specified, will send the output to a printer.

Keywords related to writing, editing, modifying and displaying programs

Disk and File Commands

DEVICE *devnum*	Sets the device number for the default device.
DIR *[num]*	Lists the current device's directory. If num is supplied, will be sent to a printer
DISK *["command"[, devnum]]*	Equivalent to OPEN 15,devnum,15,"command":CLOSE 15. Uses default device if not specified. Initializes the device if command not specified.

Disk and File Commands (cont)

LOAD *"filename"[, devnum]*	Load a file from the default device. If filename not specified, will use last specified filename.
SAVE *"filename"[, devnum]*	Save a file to the default device. Filename must be 12 characters or less. If filename not specified, will use last specified filename.
VERIFY *"filename"[, devnum]*	Verify the program in memory against a file on the default device. If filename not specified, will use last specified filename.

Commands related to using disk drives

Variables

variable$	At the end of a variable name, a $ specifies that the variable is a STRING.
CLR	Clears the memory used by all variables.
DECIMAL *variable[, variable[, variable[, ...]]]*	Creates new decimal variables.
DIM [DECIMAL] *variable(value)*	Create array *variable* of *value* size.
GLOBAL	Restores the global variable scope
LET	Assigns a simple value to a simple variable. Useful for speed.
LOCAL	Starts a local scope for variables.
TAG *tag* = *value*	Creates a TAG named *tag* with the value *value*. Like a label in assembler.
VARIABLES *[address]*	Moves the program variable table to *address* or 32768 if not specified.

Variables **must** start with a letter, and can be up to 8 characters long.

Numbers and the symbols !, @, #, %, & and ? are allowed in variable names.

Anything past 8 characters is silently ignored.

Variable names can *contain* keywords, but cannot start with keywords.

Strings must be terminated with a dollar ($) sign.

All variables are assumed to be integer variables by default.

Tags cannot be to the left of an equals sign in a math expression.

Math

vov AND *vov*	Performs a logical (bitwise) AND.
vov OR *vov*	Performs a logical (bitwise) OR.
vov EOR *vov*	Performs a logical (bitwise) Exclusive OR.
ABS(*vov*)	Returns the absolute value of *vov*.
INT(*vov*)	Returns the integer value of *vov*, rounded down.
SGN(*vov*)	Returns the sign of *vov*.
WHOLE(*vov*)	Returns the integer value of *vov* without rounding.
FRAC(*vov*)	Returns the fractional value of *vov*, stripped of the sign.
ABS(*vov*)	Returns the absolute value of *vov*.
RANDOM	Initializes the random number table
RND	Generate a random number
π	The value of PI in decimal.

Parentheses are **not** allowed in mathematical expressions.
e.g. A=4*(3+4) is not allowed and would need to be expressed as A=3+4:A=A*4

Order of operations is **not** followed. All expressions are strictly evaluated from left to right.
e.g. 4+3*5-2*6 would be evaluated as 4 + 3 = 7, 7 * 5 = 35, 35 - 2 = 33, 33 * 6 = 198.

The functions **USR**, **FRE**, **POS**, **SQR**, **LOG**, **EXP**, **COS**, **SIN**, **TAN** and **ATN** are not available.

Speedy Math

ADD *vop* = *vov* + *vov*	
COMPARE *vov*, *vov*	Both parameters must be 2 byte ints.
DEC *vop*	Decrements *vop* by 1.
DOUBLE *vop*	Multiplies *vop* by 2.
HALF *vop*	Divides *vop* by 2.
INC *vop*	Increments *vop* by 1.
SUBTRACT *vop* = *vov* - *vov*	

These commands only with with non-arrayed *integer* variables, tags and pointers, except **INC**, **DEC**, **HALF** and **DOUBLE** which also work on non-arrayed *decimal* variables.

Bitmap Commands

BITMAP *[bmp, multicolor, map, drawto, screen, color1, color2, color3, clearcol, clear map]*	Turns modes *bmp* and *multicolor* on (1) or off (0).

Bitmap Commands (cont)

BMPCLR *[clear-map[, clearcol]]*	Clears the currently visible bitmap screen if *clearmap* is 1, and color screen if *clearcol* is set to 1. If neither argument is specified, will clear both.
BMPCOL *screen, color1, color2, color3, clearcol[, clearmap]*	Sets the bitmap colors that will be used, and defines which screen to use. If clearmap is added and set to 1, will clear the bitmap.
BMPLOC *map, drawto*	*map* sets with bitmap screen will be visible (0-7). *drawto* sets with screen will be drawn to with the drawing commands.
HLINE *x, y, len, color*	Draws a straight, horizontal line starting at (*x*, *y*) and continuing to the right for *len* pixels with color *color*
LIMITS *width, height, x-pos, y-pos, colorplot*	Limits the area on the bitmap that drawing commands will affect.
LINE *x1, y1[, x2, y2[, color]]*	Draws a line on the bitmap from (*x1*, *y1*) to (*x2*, *y2*), in color *color* if specified. If *x2* and *y2* are not specified, will draw a line from (*x1*, *y1*) to the current coordinate.
PLOT *x, y, color*	Draws a pixel on the bitmap at coordinates (*x*, *y*) in color *color*.
VLINE *x, y, len, color*	Draws a straight, vertical line starting at (*x*, *y*) and going down for *len* pixels with color *color*

All parameters can be either values or variables. If a parameter is left off, a default or previous value will be used.

Sprite Commands

CODE *values....*	Any code following this command will be stored in memory at the location indicated by the "code" pointer.
COLLISION *selection*	Copies the collision registers and zeros the copied register. If *selection* is 0, copies the sprite-to-sprite register, if 1 copies the sprite-to-foreground register.
DETECT *mob#[, mob#[, mob#[, ...]]]*	Used after the COLLISION command. Checks is the specified sprites were involved in a collision.
MOB *number, on, multicolor, priority, x, y, x-add, y-add*	Chooses which sprite to make current. Initializes the sprite. *number* chooses which sprite (0-7). *on* turns it on or off, *multicolor* enables or disables multicolor mode, *priority* enables or disables background priority, *x*, *y* choose the initial coordinates, *x-add* and *y-add* set the offsets.
MOBCLR	Clears all of the sprite registers. Recommended to use at the beginning of your program if you're using sprites, and also when you want to clear the screen of sprites.
MOBCOL *color, shared1, shared2*	Sets the sprite colors. *color* sets the color of the current sprite, *shared1* sets the first color shared by all multicolor sprites, *shared2* sets the second.
MOBEXP *x-expan, y-expan*	Enables and disables the *x-expan*sion and the *y-expan*sion of the current sprite.

Sprite Commands (cont)

MOBPAT *shape#, bank*	Moves the **CODE** pointer to point at the specified sprite she's data. *shape#* specifies the shape to point at, *bank* specifies the bank where the coded data will be sent.
MOBSET *shape#, number, number, number, ...*	Initialize a sprite from The Spreditor.
MOBXY *x, y, x-add, y-add*	Moves the current sprite to the coordinates (*x*, *y*). *x-add* and *y-add* set the offsets.
SHAPE *byte[, byte[, byte[, ...]]]*	Changes the current sprite's shape. IF more than 1 shape specified, will set the shape for the following sprites.

Commands begin with MOB as a carryover from Simon's BASIC.

Interrupt Commands

HALTINT	Stops the interrupt totally, returning interrupts to normal.
INTEND *flag*	Should be the last statement in your interrupt routine. If *flag* is 0, JMP to BASIC's hardware timer routine, 1 will RTI.
INTERRUPT *raster, line#*	Create a new raster interrupt at line *raster* (50-249) which calls the code at *line#*.
RASTER *raster*	Selects the next raster line to interrupt.
STARTINT	Should be the first command in your interrupt routine.

Lets you program raster interrupts.

It is critical to halt your interrupts before exiting your program.

Using Machine Language

Branching and Jumping	Simply put the line number after the branch or jump command, e.g. `JMP1000`
Tags and variables	Can be used in place of values and addresses

Using Machine Language (cont)

ML-safe commands	**START, GOTO, GOSUB, RETURN., REM, TAG, PROC, MODULE, LOCAL, GLOBAL, ADD< SUBTRACT, COMPARE, HALF, DOUBLE, VARIABLES, HALT, RESUME, VERSION, DEBUG, STARTINT, RASTER, BYTES, STRINGS**
START *address*	Specifies the starting location for an ML program. Must be placed at the vey beginning on your ML program, and only used once.

Mnemonics must be enclosed in [] brackets, e.g.

```
100 [LDA1: ORA #1: STA1]; TURN BASIC ROM ON
```

A semicolon (;) starts a comment in ML mode.

BASIC Keywords

ASC(*string*)	Returns the ASCII value of *string*
BUTTON *joynum*	Returns 1 if the joystick button is pressed, 0 if not.
BYTES *count[, byte[, tag[, alignment]]]*	When compiling, will insert *count* bytes of value *byte* (default 0), with a (string) label of *tag*, aligned to *alignment*.
CHR$(*vov*[, *count*])	Appends ASCII character *vov* to a string, 1 or *count* times.
CLOCK *[jiffies]*	Sets the CLOCK to *jiffies* if specified, or 0 if not.
CLOSE *file#, file#, ...*	Close 1 or more files.
CLS *[pokecode[, color]]*	Clears the current text screen. Uses space if *pokecode* not specified. Colors will not be changed unless *color* is specified.
CMD *file#[, string]*	Redirects all I/O to file *file#*. Optionally sends *string* to the file.
COPY *start, end, new*	Copies the memory from addresses *start* - *end* to address *new*
DATA *val, val, ...*	Hold data to be READ later.

BASIC Keywords (cont)

DEBUG *0 / 1*	Enable (1) or disable (0) DEBUG mode, which reduces the number of passes for compilation. Will result in slower and larger programs.
DETEXT(*type*)	Returns how much extended memory of *type* is attached to the system.
DO *line#, times*	Run line *line# times* times.
DUP$(*string*, *count*)	Duplicates string *string count* times.
ELSE *statement*	If the prior IF *expression* evaluated to FALSE, *statement* will be executed.
END	End the execution of the program and returns screen to normal.
FETCH *count, destination, reu[, bank]*	Copies *count* bytes from an attached REU at address *reu* and bank *bank* to C64 address *destination*
FILL *start, end[, byte[, step]]*	Fills the memory from address *start* to address *end* with value *byte* (default 0) incrementing the address by *step* (default 1)
FOR *var=start* TO *end* [STEP *val*]	Defines a FOR loop that iterates variable *var* from the value *start* to the value *end*, defaulting to incrementing by 1 if STEP is not defined, or by *val* if it is defined.
GET *variable*	Read a character and put in it *variable*
GET# *file#*, *variable*	Read a character from *file#* and put in it *variable*
GOSUB *line#[, line#[, ...]]*	Runs a subroutine at *line#*. If more than 1 line# is specified, will run each one in the order they are specified.
GOTO *tag* / *line_number*	Jumps to the *line_number* or *tag* in the program

BASIC Keywords (cont)

HALT	Will stop compilation at this point. All previous code will be compiled.
IF *expression* [AND | OR | EOR *expression*]	Evaluates the *expression* and sets a flag that will be acted upon when the program reaches a THEN statement.
INPUT *var, var, ...*	Read lines and put the values in *var*.
INPUT# *file#, var[, var[, ...]]*	Read lines from file *file#* and store them in *var*
JOIN *vop* = *low, high*	The opposite of SPLIT
JOY(*joynum*)	Returns the value of the joystick port *joynum*. *joynum* is typically 1 or 2.
KEYPRESS *[vov[, vov]]*	If *vov* not specified, will wait for **any** keypress, otherwise will wait for *vov* to be pressed. If a second *vov* is specified will act like an IF block and will be FALSE for first char or TRUE for second char.
LEFT$(*string*, *count*)	Returns *count* characters from the left of *string*
LEN(*string*)	Returns the length of *string*.
LOC(*x, y*)	Moves the cursor to location *x, y* on the current text screen.
LONGPEEK(*address*)	Will return a single value from *address* on a SuperCPU.
LONGPOKE *address, val, val, val, ...*	Will poke into the extended memory of a SuperCPU.
MID$(*string*, *position*, *count*)	Returns *count* characters, starting at index *position* from *string*.

BASIC Keywords (cont)

MODULE *filename*[, devnum[, address]]*	When compiling, write this section to separate module file *filename* for reusability. The default device will be used if not specified. Address 49152 will be used for loading if not specified.
MODULE END	The end of the module to be written
NEXT *var[, var[, ...]]*	The end of the FOR loop. *var* must match the FOR loop you are continuing.
ON *var* GOSUB | GOTO *line#, line#, ...*	Will jump to *line#* that matches the value of *var*.
OPEN *file#, dev#, secondary, string*	Open a connection to device *dev#* assigning it to file *file#* with a secondary parameter of *secondary* and send *string* through the open file.
PADBUT *joynum*	Returns 1 if the paddle button is pressed, 0 if not.
PADDLE *joynum*	Returns the value of the paddle, 0-255. Paddles 1-2 are in *joynum* 1, 3-4 are in 2.
PAUSE *seconds[, jiffies]*	Will pause execution for *seconds* seconds. If an optional *jiffies* is specified, will pause for an additional (*jiffies*/60)s.
PEEK(*vov[, index]*)	Will return the memory at address *vov*, optionally offset by *index*.
POKE *address*, *vov, vov, vov, ...*	Will put values *vov* in consecutive memory starting at *address*. Can also be used with strings.
PRINT *expression*	Prints *expression* to the current text screen
PRINT# *file#, expression*	Print *expresion* to file *file#*
READ *vop, vop, ...*	Read values from a DATA statement

BASIC Keywords (cont)

REM	Turns the rest of the line into a comment.
RESTORE *[line#]*	Resets the pointer to the start of all DATA statements, or to the DATA statement on line *line#*
RESUME	Will resume compilation after a HALT. **Must** be at the beginning of a line, or it will be ignored.
RETURN	Ends a subroutine and sends program flow back to the GOSUB statement.
REUPEEK(*address, bank*)	Will return the value from an attached REU at *address* in bank *bank*
REUPOKE *address, bank, val, val, val, ...*	Will write the values to an attached REU starting at *address* in bank *bank*
RIGHT$(*string*, *count*)	Returns *count* characters from the right of *string*
SPC(*vov*)	Prints *vov* spaces
SPLIT *low, high[, high2]* = *vov*	Splits a variable into low and high bytes.
STASH *count, address, reu[, bank]*	Copies *count* C64 memory bytes at address *address* to the attached REU address *reu* in bank *bank*
STATUS	Reads and clears the STatus
STOP	Stops the program execution but does not reset the screen.
STR$(*vov*)	Converts number *vov* into a string.
STRINGS *[size]*	With no parameter, stretches the string field to 53247, otherwise to *size*.
SWAP *count, c64, reu[, bank]*	Swaps the main memory at address *c64* with the memory on the attached REU at address *reu* in bank *bank*

BASIC Keywords (cont)

SWITCH *start, end, start2*	Swaps the memory at addresses *start* - *end* with the memory starting at address *start2*
SYS *address[, A, X, Y, ST]*	Starts execution of ML code at *address*. If the A, X, Y and ST values are specified, they will be loaded into the registers before starting.
TAB(*vov*)	Moves the cursor to *vov* on the current line.
THEN *statement*	If the prior IF *expression* evaluated to TRUE, *statement* will be executed.
TRAP *line# [, vop]*	Sends control of your program to *line#* on error. *vop* if specified must be a non-arrayed int which will have the address of the error stored in it.
VAL(*string*)	Returns the mathematical value of *string*.
VERSION *number*	Specifies which version of Vision BASIC needed to compile the block of code.
WAIT *address, and, eor*	Will wait for a non-0 result from PEEKing *address* and filtering with AND *and* and EOR *eor*

FOR-TO-STEP-NEXT loops and DO loops will only work on integer variables.

Functions and subroutines

POINT *vop = line#*	Sets *vop* to the address of the compiled code for line *line#*.
POINT TAG *tag = line#*	Creates a tag and points it to the address of the compiled code for line *line#*.
PROC *tag[. vop[, vop[, ...]]]*	Defines the start of a subroutine named *tag* with parameters *vop*.
PASS *vop[, vop[, vop[, ...]]]*	Defines paramaters *vop* for a subroutine. Must be the first command after PROC if you're passing parameters.

Functions and subroutines (cont)

RETURN	End the subroutine and send execution back to the calling line.
SEND *vov*	Will make the subroutine return the value *vov*. This must be the final command before RETURN if used.
TAG *tag[= vov]*	Creates a TAG named *tag*. If *vov* is not specified, tag will get the current address in the program. Used like a LABEL in assembler.

Call a subroutine like

```
tag.vo p,v op,vop
```

Strings and string variables cannot be returned from subroutines.

Sound Commands

ADSR *attack, decay, sustain, release*	Specifies the *attack*, *decay*, *sustain* and *release* parameters for the current voice.
CUTOFF *freq*	Sets the cutoff frequency to *freq* for the SID filtering system.
FILTER *voice1, voice2, voice3, ext, resonance*	Enables or disables the filters of *voice1*, *voice2*, *voice3*, the output of the *external* input and the *resonance* value.
FREQ *freq*	Specifies that the current voice will play frequency *freq*.
PULSE *width*	Specifies the pulse waveform width for the current voice.
SIDCLR	Clears the sound registers
VOICE *num*	Chooses which voice will be used (1-3)
VOL *volume, low, band, high, disconnect*	Controls the main volume and filter selection. Can enable or disable the *low*, *band* and *high* pass filters. If *disconnect* is enables, disconnects the output of voice 3.

Sound Commands (cont)

WAVE *gate, wave, ring, sync test*	Enables or disables the *gate*. *wave* is 1 (triangle), 2 (sawtooth), 4 (pulse) or 8 (noise). *ring* chooses ring modulate oscillators. *sync* chooses sync modulate oscillators. *test* enables or disables the voice oscillator.

All parameters can be values or variables. If a parameter is left off, a previously used value will be used, or a 0 if no value has been specified before.

The commands **FREQ**, **PULSE**, **ADSR**, and **WAVE** require you to set a current **VOICE** before calling them.

0 disables, 1 enables

Text Video Commands

BANK *bank*	Selects the active 16K memory bank for video. This is **not** for setting REU banks. You will probably not need to ever use this command.
BLANK *blank[, off]*	Blanks or restores the screen. *off* turns the video chip off completely.
CHARPAT *character, charset*	Moves the "code" pointer to point at a specific character image. *character* is the character that you want to point at, *charset* is the character set that the character is in.
CHARSET *charset*	Selects the desired character set.
COLORS *text, border, screen, color1, color2, color3*	Sets the color registers.
COPYSET *charset[, case]*	Copies the C64 character set to location *charset*. If *case* is 0, copy uppercase, if 1 copy lowercase.
EXTENDED *on[, color1, color2, color3]*	Turns on or off extended color mode. If colors are supplied, will set the 3 background colors.

Text Video Commands (cont)

LOWERCASE *[disable]*	Changes the character set to lowercase. If *disable* is 1, disables keyboard toggling between upper and lower case.
MULTI *on[, color1, color2]*	Turns on or off multicolor mode. If colors are specified, will set the background colors.
NORMAL	Resets the screen to normal text mode.
UPPERCASE *[disable]*	Changes the character set to uppercase. If *disable* is 1, disables keyboard toggling between upper and lower case.
PANX *panvalue, columns*	Pans the screen horizontally. *panvalue* can be 0-7 (0=none), if *columns* is 0, sets 38 column screen, 1 sets 40 column screen.
PANY *panvalue, rows*	Pans the screen vertically. *panvalue* can be 0-7 (3=none). If *rows* is 0, 24 row screen, if 1, 25 row screen.
VIDLOC *screen, printto, charset*	Moves the text screen to and of the 64 1K screens available. *screen* chooses which 1K to use, *printto* chooses which screen to print to (you will probably want this to be equal to *screen*), *charset* selects to location of the character set.