SEMBERIA C64 BASIC EXTENSION

Introduction

There has been a lot of talk about the shortcomings of C64 BASIC in the more than 40 years of its existence. Designed in 1977, BASIC for the Commodore PET was the best in its class, offering commands for working with alphanumeric variables and real numbers with transcendent functions. But BASIC from the era when teleprinters were still a relevant input/output device remained in that form for the next seven years on later Commodore models, without even such rudimentary commands to clear the screen, position the cursor on the screen, or activate high-resolution graphics, so PEEK, POKE and CHR$ were used. Although this did not prevent the Commodore 64 from becoming the most successful home computer, many computers with weaker hardware than Commodore 64 (Sinclair ZX Spectrum, MSX 1, Atari 400, Apple II+), were much more convenient to program, because they had the necessary BASIC commands. Unbelievably, the debate over which computer was better continues to this day.

Extensions of C64 BASIC appeared quite early, the most famous of which are Simons BASIC and BASIC 3.5. In addition, computer magazines have published listings of how to extend BASIC with new commands or machine/assembly routines that provide hardware access. However, these extensions are not without their drawbacks:

- They need to be always pre-loaded into memory before using new commands, even for finished BASIC programs, which makes preparing for work more difficult and time-consuming.

- Some of them significantly reduce the space provided for BASIC programs (Simons BASIC for 8K).

- Different extensions often occupy the same memory locations, so we cannot e.g. use an extension that adds graphics commands along with the RENUMBER routine because they are both assembled from address 49152

- We can't choose which commands we want to use and which ones we don't want to use, so even the ones we don't need take up memory.

- Programs written in extended BASIC can only be used by those who have those extensions

SEMBERIA

SEMBERIA (Simple Embeddable Modular Basic Extension Relocatable It's Acronym) is an extension of BASIC that attempts to overcome these limitations. It is not as ambitious as Simons BASIC. In its full configuration, it has 27 new commands and takes up almost 2 kilobytes of RAM. However, all commands are relocatible and mostly mutually independent. Therefore, it is possible to place the BASIC extension in almost any memory location:

- From location 49152, leaving all 38 kilobytes of BASIC space intact, but also 1 kilobyte within the machine program area for other machine language programs or sprites (HIGH version)

- From location 2061, which reduces the BASIC space to 36 kilobytes, but allows BASIC programs using the new commands to be saved together with the extension and loaded and started with the usual LOAD and RUN commands, without the need for any loader. Programs can be edited later.

- In BASIC DATA lines that makes it possible to customize the BASIC extension (choosing memory location and commands to include or exclude), and then save as machine code block.

- Between BASIC programs and variables, where machine programs are usually placed on the C64. It also allows saving BASIC programs together with this extension, but not program editing. This would be used with programs initially developed using classic V2 BASIC, with additional commands later used when building the final version.

The author of the SEMBERIA extension is prof. Samir Ribić, Ph.D. from Sarajevo, Bosnia and Herzegovina. Yes, Semberija is a name for north-east part of Bosnia and Herzegovina, although I live in the central part. It was developed as a retro project to celebrate 40 years of programming and it is completely free to use and modify. The motivation of the author, otherwise oriented all his life towards ZX Spectrum and IBM PC compatible computers, is to show what should have existed in 1984 to start his programming life with the (then hated) C64.

Starting BASIC extension SEMBERIA

SEMBERIA can be downloaded from the Internet in the form of a diskette file (D64) or a tape files (TAP, WAV). D64 and TAP files will probably be used through an emulator (such as VICE). WAV is intended to be saved to tape recorder using PC sound card and then loaded on a real C64 using 1530 tape device. Each of them contains three versions: HIGH, LOW and INSTALL.

Version HIGH, location 49152

This version does not take memory reserved for BASIC programs, but needs to be loaded before the BASIC program that uses new commands. Run the following commands to load and activate it.

`LOAD "HIGH",8`	for disk version
`LOAD "HIGH"`	for tape version

Start the program with

```
RUN
```

It relocates SEMBERIA to location 49152, initializes it and delete BASIC program. After that, new commands are available, and you can use them in your BASIC pograms. If for some reason the BASIC extension needs to be restarted, it can be done with SYS49152

You can save programs written in extended BASIC using the commands:

```
SAVE "NAME"                on the tape
SAVE "NAME",8              on diskette
```

Before loading a program written in extended BASIC, you should load and run the HIGH version of the SEMBERIA extension, as described above, and then execute

```
LOAD "NAME"                from the tape
LOAD "NAME",8              from diskette
```

LOW version, immediately before the BASIC program

This version takes two kilobytes of BASIC memory, but can be saved together with BASIC program that uses new commands, becoming an integral part of it. Run the following commands to load:

```
LOAD "LOW",8               for disk version
LOAD "LOW",1               for tape version
```

Run the program with

```
RUN
```

After that, new commands are available.

You can save programs written in extended BASIC as follows

```
POKE 44,8
SAVE "NAME"                on the tape
RUN

POKE 44,8
SAVE "NAME",8              on diskette
RUN
```

If for some reason the BASIC extension needs to be restarted, it can be done with SYS2061

INSTALL version in DATA lines

This version is used to customize SEMBERIA BASIC without recompiling source code. Run the following commands to load and activate

```
LOAD "INSTALL",8           for disk version

LOAD "INSTALL"             for the tape version
```

The BASIC extension is stored in DATA lines. It is possible to edit the listing in order to add or delete commands, change the default loading address or filename. See the customizing section of this manual for more info.

Run the program with

```
RUN
```

Wait until the DATA lines are read and the customized version of SEMBERIA saved to disk or (if you have modified it) tape. Reset your computer, and assuming that you have not modified INSTALL version (keeping the default options for name and start location), load and initialize SEMBERIA BASIC using

```
LOAD "EXBAS",8,1
SYS 49152
NEW
```

New commands

In order to minimize memory usage when recognizing commands and to make them faster, all new commands start with an exclamation mark followed by two bytes represented by letters or existing BASIC tokens.

Textual screen related commands

!AT x,y

Sets the location of the text cursor, column x (0 ≤ x ≤ 39), row y (0 ≤ y ≤ 24). The command works in standard text and user definable mode. The following example prints the text in the column ten, row eight.

```
10 !AT 10,8 : PRINT "HELLO WORLD"
```

!BCLR background,border

Sets the background and border color of the screen for standard text, user definable, and multi-color graphics modes. The first parameter (background) represents the background color, and the second parameter (border) represents the border color. The values of both parameters are integers, 0 ≤ background ≤ 15, 0 ≤ border ≤ 15, and they represent the following colors:

0	black	8	orange
1	white	9	brown
2	red	10	pink
3	cyan	11	dark grey
4	purple	12	grey
5	green	13	light green
6	blue	14	light blue
7	yellow	15	light grey

If the value of the parameter is between 16 and 255, it is taken as if the entered value is the remainder when the parameter is divided by 16. Values outside the range between 0 and 255 cause an Illegal quantity error.

Example: Sets red background, yellow border.

```
!BCLR 2,7
```

!TCLR color

Sets the text color for standard text and user definable text modes. The parameter value represents the color code described in the BCLR command.

Example: The following program switches the screen to white letters on a black background:

```
10 !BCLR 0,0 : !TCLR 1
```

!CLRS

Clears a standard text display or a text display that allows user-defined characters and places the cursor in the upper left corner.

Example:
```
!CLRS
```

!ORD

It switches to standard text mode (40x25), with video memory located at address $0400 (1024).

Example:
```
!ORD
```

!USRT

It switches to text mode (40x25), which allows for character predefining, with video memory located at $CC00 (52224) and character definition at $E000 (57344). Each character in the definition occupies 8 bytes, ordered by display codes.

Example:
```
!USRT
```

!CDEF addressorpattern

If the parameter is an integer, it sets the starting address of the character we want to predefine. The description of the character with screen code n starts from the address 57344+8*n. For example, the letter "K" has a screen code of 11, so its description starts at address 57432. Each character consists of 8 lines represented by an eight-bit binary number. The starting address is written to location $FB-$FC (251 and 252) and as other SEMBERIA commands use this location, the lines describing the character shape must be immediately after the line setting the starting address.

If the parameter is a pattern representing a string of eight characters “.” and "#" it generates the next byte of the character shape and increases the position by 1. The dot represents the binary zero, i.e. the point in the character that will be the background color, and the hash character the binary one, i.e. point in the character foreground color.

Example: This program switches to user text mode and presets the letter K into a little man character, and animates it in low resolution.

```
10 !USRT
20 !CLRS
30 !CDEF 57432
40 !CDEF ...###..
50 !CDEF ...###..
60 !CDEF ....#...
70 !CDEF ########
80 !CDEF .. ###..
90 !CDEF ...###.
100!CDEF ...#.#..
110!CDEF ..##.##.
120 FOR I=0 TO 38
130 !AT I,0: PRINT " K"
140 !WAITFOR 100
150 NEXT I
160 GOTO 120
```

High resolution and multicolor graphic commands

!GR mode

Activates graphics mode.

The mode parameter selects the type of operating mode. A value of mode=0 activates a 320x200 resolution mode with two colors within an 8x8 pixel square. A value of mode=1 activates a 160x200 resolution mode with four colors within an 8x8 pixel square. In both modes, the pixel in the upper left corner of the screen is at coordinates (0,0) and in the lower right corner at coordinates (319,199). Therefore, for the 160x200 mode the coordinates X,Y and X+1,Y for even values of X represent the same pixel, while for the 320x200 mode they represent two different pixels. The video memory is located at address $E000 (57344) and occupies 8000 bytes, under the KERNAL ROM. Attribute squares that define 8x8 color areas are located at the address $CC00 (52224) and occupy 1000 bytes. When switching to these modes, the contents of the screen and color memory are not deleted, so it is recommended to call the !CLRG command before switching to the graphics mode for the first time.

Example: The example initially switches to 320x200 mode without clearing the contents of video memory, then clears it and re-enters 320x200 mode, then skips to 160x200 mode and returns to the odrinary text mode. Press RETURN between each transition

```
10 !GR 0
20 INPUT A$
30 !CLRG 7,0
40 !GR 0
50 INPUT A$
60 !GR 1
70 INPUT A$
80 !ORD
90 INPUT A$
```

!CLRG color1,color2

Clears the graphics screen and sets the initial colors in all color squares. The parameter values represent the color label described in the !BCLR command.

For 320x200 graphics mode, the first parameter (color1) represents the background color, i.e. the color that will be used when the last parameter of the commands !LI and !PPOS is equal to 0, and the second parameter (color2) the color of points and lines for all squares with colors, i.e. the color to be used when the last parameter of the !LI and !PPOS commands is equal to 1.

For 160x200 graphics mode, the first parameter (color1) represents the color to be used when the last parameter of the !LI and !PPOS commands is equal to 3 , and the second parameter (color2) represents the color to be used when the last parameter of the !LI and !PPOS equal to 2.

Example: It sets the graphics mode to 320x200 and prepares drawing with black pixels on a yellow background and draws a line of the selected black color.

```
10 !GR 0
20 !CLRG 7,0
30 !LI 0,0,100,100,1
```

!GETPRINT over

Copies the contents of the ordinary text screen to the graphics video memory, using the character set in ROM. The over parameter represents a way of characet copying:

0 copy uppercase – replaces the text by deleting the already existing high-resolution image, using a character set that uses uppercase letters and graphic characters

1 over uppercase – merges the text with a high-resolution image, using a character set that uses uppercase letters and graphic characters

2 copy lowercase – replaces the text by deleting the already existing high-resolution image, using a character set that uses both lowercase and uppercase letters

3 over lowercase - merges the text with a high-resolution image, using a character set that uses both lowercase and uppercase letters

In both graphic modes (320x200 and 160x200) the resulting text is such that 40 characters fit in a row, but in the 160x200 mode it looks a bit ugly.

Example: It sets the graphics mode to 320x200, draws a diagonal line, prints HELLO WORLD on the text screen and then merges the text screen image to the existing graphics.

```
10 !GR 0
20 !CLRG 7,0
30 !LI 0,0,319,199,1
40 !CLRS
50 !AT 10,7: PRINT "HELLO WORLD"
60 !GETPRINT 1
```

!PPOS x,y,c

Plots a pixel on the screen at (x,y) coordinates in graphics mode. In both modes (320x200 and 160x200) the upper left corner is at coordinates (0,0), and the lower right corner is at coordinates (319,199).

The parameter c represents the color from the palette. In 320x200 mode if c=0, it represents the background color, and if c=1 it represents the drawing color. These colors are set with the command !CLRG or !SQRCLR. In 160x200 mode, if c=0, it represents the background color (see !BCLR), c=1 is the text color (see !TCLR), for c=2 the point is drawn with the color that is the second parameter of the !CLRG command, and for c=3 with color which is the first parameter of the command !CLRG.

Example: Sets graphics mode to 320x200, black on green, draws a sine wave in black, then clears it.

```
10 !GR 0
20 !CLRG 5,0
30 FOR X=0 TO 319
40 !PPOS X,100+100*SIN(X/20),1
50 NEXT X
60 FOR X=0 TO 319
70 !PPOS X,100+100*SIN(X/20),0
80 NEXT X
90 !ORD
```

Example: Sets the graphics mode to 160x200, with a blue background, setting yellow and red for palette colors 2 and 3, blue for the background color, and purple for the text color and palette color number 1. After that, parallel horizontal lines are drawn with a loop in palette colors 1 , 2, 3, which are assigned in that order to the physical colors purple (4), red (2), and yellow (7).

```
10 !GR 1
20 !BCLR 6,6: !CLRG 7,2: !TCLR 4 : !CLRS
30 FOR X=0 TO 319
40 !PPOS X,80,1
50 !PPOS X,90,2
60 !PPOS X,100,3
70 NEXT X
```

!LI x1,y1,x2,y2,c

Draws a straight line between the points with coordinates (x1,y1) and (x2,y2) using the logical color specified by the c parameter. In both graphic modes (0≤x1≤ 319), (0≤y1≤199), (0≤x2≤319), (0≤y2≤199) is valid. The meaning of the c parameter is the same as that of the !PPOS command.

Example: Turns on 320x200 graphics mode, green on black and draws two diagonal lines

```
10 !GR 0
20 !CLRG 5,0
30 !LI0,0,319,199,1
40 !LI0,199,319,0,1
```

!SQRCLR x,y,color1,color2, color3

Sets the color of the 8x8 attribute square associated with the graphic image with a resolution of 320x200 or 160x200.

The coordinates of the squares are column x (0≤x≤39), row y (0≤y≤24). Each square defines colors of the pixels at position (x1,y1) where x*8 ≤ x1 ≤ x*8+7, y*8 ≤ y1 ≤ y*8+7 .

The relationship between colors given as the last parameter of !LI and !PPOS commands and parameters color1, color2 and color3 used in !SQRCLR is given in the following table.

Last parameter of !LI and !PPOS	Mode 320x200	Mode 160x200
0	color1	background color
1	color2	color3
2	n/a	color2
3	n/a	color1

Example: Turns on the 320x200 graphics mode, green on black and draws two diagonal lines, and in the center of the screen in two squares changes the background to yellow and the print to gray.

```
10 !GR 0
20 !CLRG 5,0
30 !LI0,0,319,199,1
40 !LI0,199,319,0,1
50 !SQRCLR 19,12,7,15,0
60 !SQRCLR 20,12,7,15,0
```

Sprite commands

!SDEF num,addr,color

Specifies the memory address of the sprite definition and the initial color.

The parameter num (0≤num≤7) specifies the sprite number.

The addr parameter represents the memory address of the sprite bit pattern definition. The specified address should be divisible by 64 and in the range 0-16384 for plain text mode, or in the range 49152-65535 for graphic and user-defined text mode. To avoid conflicts with the BASIC program, safe locations for plain text mode are 832, 896 and 960. If graphics mode is active and SEMBERIA is located at location 49152, safe locations are 51200, 51264, 51328, ..., 52160 , 65344 and 65408. In user-defined text mode, in addition to the locations safe for graphics mode, locations 61440, 61504, 61568, ..., 65280 are also safe. If SEMBERIA is placed before or after the BASIC program, in graphics and user-defined in text mode, the locations 49152, 49216, 49280, ... 51136 can also be used for the sprite description. The sprite has dimensions of 24x21 pixels and the sprite description occupies 63 bytes.

Each line of a sprite is represented by three adjacent bytes represented in binary. An easier way to define a sprite is with the !CDEF command.

The color parameter (0≤color≤15) represents the color of the sprite in two-color mode.

Example: Switches to user text mode and defines a striped pattern for the yellow sprite shape, which it then displays. Decimal value 170 is 10101010 in binary.

```
10 !USRT
20 FOR I=51200 TO 51263
30 POKE I,170
40 NEXT I
50 !SDEF 1,51200,7
60 !SPOS 1,200,100,1
```

!SPOS num,x,y,visible

Sets the position of the sprite and makes it visible or invisible.

The parameter num (0≤num≤7) specifies the sprite number. The parameters x and y determine the coordinates of the upper left corner of the sprite (0≤x≤511, 0≤y≤255). The coordinates are different from the graphics and text screen coordinates in order to display the sprite part on the screen as well. The visible parameter can be 0 or 1 and determines whether the sprite will be displayed on the screen or not.

Example: Switches to graphics mode, defines a grid pattern sprite, which then moves diagonally across the screen.

```
10 !CLRG 0,1
20 !BCLR 0,0
30 !GR 0
40 FOR I=0 TO 63
50 POKE 51200+I,60
60 NEXT I
70 !SDEF 3,51200,4
80 FOR I=0 TO 511
90 !SPOS 3,I,I/2,1
100 NEXT I
```

!EXPAND num,dimension,value

Changes the sprite's visibility, height, width, background priority, and number of colors.

The parameter num (0≤num≤7) specifies the sprite number.

The dimension parameter means which sprite data is changed and it can have values as in table

Value	Name	Meaning
0	visible	whether the sprite will be displayed or not
1	double width	whether the sprite pixels will be doubled in width
2	double height	whether the sprite pixels will be doubled in height
3	priority over background	whether the sprite will be displayed in front of the background or behind backround
4	multicolor sprite	whether the sprite will be in four color mode, if this value is selected the colors are selected by the background color, the contents of locations 53285 and 53286 and the last parameter of the !SDEF command.

The value parameter has the values 0 or 1 and indicates whether the corresponding value of the dimension parameter is off or on.

Example: Defines a sprite according to the contents of text video memory and then renders it twice as wide and twice as tall.

```
10 !ORD
20 !SDEF 4,1024,4
30 !SPOS 4,100,100,1
40 !EXPAND 4,1,1
50 !EXPAND 4,2,1
```

!CDEF addressorpattern

If the parameter is an integer, it sets the starting address of the sprite we want to preset. Each sprite consists of 21 lines represented by three eight-bit binary numbers.

The starting address is written to the location $FB-$FC (251 and 252) and as other SEMBERIA commands use this location, the lines describing the character lines must be immediately after the line which sets the starting address.

If the parameter is a pattern representing a string of 24 characters “.” and "#" it writes the next line of the character description and increases the position by 3. The dot represents the binary zero, i.e. the place in the character that will be the background color, and the hash character the binary unit, i.e. place in the sign of the color of the sign itself.

For multi-color sprites, the sprite pixel color is determined by combinations of two adjacent characters in the pattern: “..”, “.#”. “#.” and “##”.

Example: Goes into graphics mode, defines a rocket shape sprite, which then rises up.

```
10 !CLRG 0,1
20 !BCLR 0,0
30 !GR 0
100!CDEF 51264
101!CDEF........................
102!CDEF.........####...........
103!CDEF........######..........
104!CDEF........######..........
105!CDEF........##..##..........
106!CDEF........##..##..........
107!CDEF........######..........
108!CDEF......##########........
109!CDEF........######..........
110!CDEF........##..##..........
111!CDEF........##..##..........
112!CDEF.......########.........
113!CDEF.......########.........
114!CDEF.......###..###.........
115!CDEF......####..####........
116!CDEF......##########........
117!CDEF......###.#.#.##........
118!CDEF......##..#.#.##........
119!CDEF.....###......###.......
120!CDEF........................
121!CDEF........................
200!SDEF 3,51264,4
210FOR I=250 TO 0 STEP -1
220!SPOS 3,200,I,1
230NEXT I
```

!IFCLOSE num

Checks if a sprite has touched another sprite or the background. If so, the least significant bit of the ST variable is set to 1. The parameter num (0≤num≤7) specifies the sprite number.

Example: Defines sprite in textual mode and checks if it colided with background or other sprite.

```
10 !ORD
20 !SDEF 4,1024,4
30 !SPOS 4,100,100,1
40 !IFCLOSE 4
50 IF ST AND 1 THEN PRINT "CRASH"
```

Memory commands

!MPOKE fromaddr,toaddr,size

Copies part of the memory starting from the address specified by the fromaddr parameter, to the address specified in the toaddr parameter, transferring the total bytes specified by the size parameter.

Example: Moves the second line of the text screen to the position of the first line

```
10 !MPOKE 1064,1024,40
```

!DPOKE addr,value

Stores the 16-bit value specified by the value parameter to two adjacent memory locations specified by the addr parameter

Example: writes the value 650 to location 40000 and then prints it

```
10 !DPOKE 40000,650
20 PRINT PEEK(40000)+256*PEEK(40001)
```

Sound commands

!NOTON

Resets the sound chip and mutes the sound. It is recommended to call this command before using other sound commands.

Example: Prepares audio chip

```
10 !NOTON
```

!VVAL volume

Sets the maximum volume for all channels. The volume parameter can have the value 0≤volume≤15. In addition to the master volume, the sustain level for all three channels is also set to the same value.

Example: Prepares audio chip and sets master volume to maximum

```
10 !NOTON:!VVAL 15
```

!NOTE channel,frequence

Plays a note using triangular wave on the specified channel. The value of the channel parameter is 1, 2 or 3. The frequence parameter appoximately gives frequence of the note in seventeenth of Herz. The recommended values for several note are given in the following table. Value 0 for the frequence stops the tone playing, or fades for the amount of time specified in release parameter of envelope (!LENV command) for this channel.

Note	Oct	Value	Note	Oct	Value	Note	Oct	Value	Note	Oct	Value	Note	Oct	Value	Note	Oct	Value	Note	Oct	Value	Note	Oct	Value
C	0	278	C	1	536	C	2	1072	C	3	2145	C	4	4291	C	5	8583	C	6	17167	C	7	34334
C#	0	284	C#	1	568	C#	2	1136	C#	3	2273	C#	4	4547	C#	5	9094	C#	6	18188	C#	7	36376
D	0	301	D	1	602	D	2	1204	D	3	2408	D	4	4817	D	5	9634	D	6	19269	D	7	38539
D#	0	318	D#	1	637	D#	2	1275	D#	3	2551	D#	4	5103	D#	5	10207	D#	6	20415	D#	7	40830
E	0	337	E	1	675	E	2	1351	E	3	2703	E	4	5407	E	5	10814	E	6	21629	E	7	43258
F	0	358	F	1	716	F	2	1432	F	3	2864	F	4	5728	F	5	11457	F	6	22915	F	7	45830
F#	0	379	F#	1	758	F#	2	1517	F#	3	3034	F#	4	6069	F#	5	12139	F#	6	24278	F#	7	48556
G	0	401	G	1	803	G	2	1607	G	3	3215	G	4	6430	G	5	12860	G	6	25721	G	7	51443
G#	0	425	G#	1	851	G#	2	1703	G#	3	3406	G#	4	6812	G#	5	13625	G#	6	27251	G#	7	54502
A	0	451	A	1	902	A	2	1804	A	3	3608	A	4	7217	A	5	14435	A	6	28871	A	7	57743
A#	0	477	A#	1	955	A#	2	1911	A#	3	3823	A#	4	7647	A#	5	15294	A#	6	30588	A#	7	61176
B	0	506	B	1	1012	B	2	2025	B	3	4050	B	4	8101	B	5	16203	B	6	32407	B	7	64814

Example: Plays Twinkle twinkle little star on one channel

```
10 !NOTON
20 !VVAL 15
30 READ A
40 IF A=-1 THEN STOP
50 READ B
60 !NOTE 1,A
70 !WAITFOR B
80 !NOTE 1,0
90 GOTO 30
1000 DATA 4291,400,4291,400,6430,400,6430
,400,7217,400,7217,400,6430,800
1010 DATA 5728,400,5728,400,5407,400,5407,400
,400,4817,400,4817,400,4291,800
1020 DATA -1
```

!LENV channel,attack,decay,sustain,release

Defines envelope of the sound played using !NOTE or !FORM.When the tone is started, it first grows to the maximum volume specified by the !VVAL command for the time specified by the attack parameter. After that, it drops to the level specified by the sustain parameter, and this transition lasts as long as specified by the decay parameter. The channel volume remains at the level specified by the sustain parameter, until it is stopped by the !NOTE command whose frequency parameter is equal to 0, After that, the volume in the channel decreases to level 0, for the time specified by the release parameter.

The value of the channel parameter is 1, 2 or 3. The sustain parameter can have a value of 0≤sustain≤15 and it represents the volume of the tone in that phase. The parameters attack, decay and release represent the time duration of individual phases, they also have values between 0 and 15 with the meaning as in the table.

Value	**Attack**	**Decay**	**Release**		**Value**	**Attack**	**Decay**	**Release**
0	2 ms	6 ms	6 ms		8	100 ms	0.3 s	0.3 s
1	8 ms	24 ms	24 ms		9	0.25 s	0.75 s	0.75 s
2	16 ms	48 ms	48 ms		10	0.5 s	1.5 s	1.5 s
3	24 ms	72 ms	72 ms		11	0.8 s	2.4 s	2.4 s
4	38 ms	114 ms	114 ms		12	1 s	3 s	3 s
5	56 ms	168 ms	168 ms		13	3 s	9 s	9 s
6	68 ms	204 ms	204 ms		14	5 s	15 s	15 s
7	80 ms	240 ms	240 ms		15	8 s	24 s	24 s

Example: The tone slowly fades due to long release parameter

```
10 !NOTON
20 !VVAL 15
30 !LENV 1,2,2,13,14
40 !NOTE 1,5738
50 !NOTE 1,0
```

!FORM channel,frequence,waveform,pulse

Plays a tone of the specified frequence, using a special waveform. Parameters channel and frequence have the same meaning like described in the command !NOTE. Value waveform determines waveform of the tone. It is generated from a binary number, with the following bit meaning:

Bit 7	Select Random Noise Waveform, 1 = On
Bit 6	Select Pulse Waveform, 1 = On
Bit 5	Select Sawtooth Waveform, 1 = On
Bit 4	Select Triangle Waveform, 1 = On
Bit 3	Test Bit: 1 = Disable Oscillator
Bit 2	Ring Modulate Osc. 3 with Osc. 2 Output, 1 = On
Bit 1	Synchronize Osc. 3 with Osc. 2 Frequency, 1 = On
Bit 0	Gate Bit: 1 = Start Att/Dec/Sus, 0 = Start Release

Some common values for waveform parameter are: 0 – note off, 17- triangular vawe, 33 – sawtooth vawe, 65 – pulse vawe, 129 – noise.

Parameter pulse (0 ≤ pulse ≤4095) is used when the vaweform is pulse. Its value is percentage of low volume in vaweform multiplied by 4095. The symetric pulse (the same length of high and low volume) is when pulse=2048. For other vaweforms this paremeter is ignored.

Example: Plays a chord using three different instruments

```
10 !NOTON
20 !VVAL 15
30 !FORM 1,4291,17,0
40 !FORM 1,5407,33,0
50 !FORM 1,6430,65,2048
60 !WAITFOR 1000
70 !NOTON
```

Program control commands

!STOPON line

In a case of program error, the execution continues from the specified BASIC program line

Should be used only inside BASIC program. Parameter line is the line number that has to exist in the program. When the error is detected, we can read the error code at memory location 781 as given in the table, and standard error handling is resumed.

	8 missing filename	16 out of memory	24 file data
1 too many files	9 illegal device number	17 undef'd statement	25 formula too complex
2 file open	10 next without for	18 bad subscript	26 can't continue
3 file not open	11 syntax	19 redim'd array	27 undef'd function
4 file not found	12 return without gosub	20 division by zero	28 verify
5 device not present	13 out of data	21 illegal direct	29 load
6 not input file	14 illegal quantity	22 type mismatch	
7 not output file	15 overflow	23 string too long	

Example: Displays reciprocals of numbers until zero is entered. After that, it is reported that error number 20 (Division by zero) has been recognized.

```
10 !STOPON 1000
20 INPUT A
30 PRINT 1/A
40 GOTO 20
1000 PRINT "ERROR "; PEEK(781)
```

!WAITFOR ms

Delays excution for a specified number of miliseconds

Example: Waits for one second

```
!WAITFOR 1000
```

!RESTORETO line

Specifies the line with the DATA command from which the subsequent READ command will take data. Parameter line is the line number that has to exist in the program.

Example: This example prints number 5, because it is at beginning of the line 120 given as parameter of !RESTORETO command.

```
10 !RESTORETO 120
20 READ A
30 PRINT A
110 DATA 1,2,3,4
120 DATA 5,6,7,8
```

Customizing

SEMBERIA is quite flexible BASIC extension. Advanced users can define new commands, delete existing ones in order to save memory or change memory address. Two principal methods for customization are using source code and using DATA lines version.

Customization using the source code

The source code of SEMBERIA BASIC is written as an assembly language textual source file. It can be compiled using the AS65 assembler and the LD65 linker that are part of the CC65 C compiler. This compiler is available for Windows and Linux based personal computers and is run from the command line.

The source code can be configured using command line parameters -D xxxx, which define additional symbols, primarily specifying the BASIC commands that will be included in the final code.

If the symbol HIGHVER is defined, then the HIGH version is compiled. The following commands assemble and link the BASIC extension on the PC, creating the high.prg file .

```
ca65 -D HIGHVER -mm near -t c64 -o high.o semberia.s
ld65 -C c64-asm.cfg -S "$0801" high.o -o high.prg
```

If the symbol LOWVER is defined then the LOW version is compiled. The following commands assemble and link the BASIC extension on the PC, creating the low.prg file

```
ca65 -D LOWVER -mm near -t c64 -o low.o semberia.s
ld65 -C c64-asm.cfg -S "$0801" low.o -o low.prg
```

If the symbol CODEVER is defined it compiles to machine code block without BASIC program appended. Special program makedata (supplied with source code of SEMBERIA) is then used to convert it into BASIC program with INSTALL version. The following commands assemble, link and convert the BASIC extension on the PC, creating the block.prg and install.prg files.

```
ca65 -D CODEVER -mm near -t c64 -o block.o semberia.s
ld65 -C c64-asm.cfg -S "$C000" block.o -o block.prg
makedata block.prg install.prg
```

By defining the symbol PARTIAL, all commands will be turned off and then we can choose which commands to embed by defining the symbols CMDxxxx, where xxxx represents the name of the command. For example, to create a TINYLOW version that will have only the !AT and !USRT commands and be only 256 bytes in size, with the properties of the LOW version, we will specify the commands:

```
ca65 -D LOWVER -D PARTIAL -D CMDAT -DCMDUSRT -mm near -t c64 -o
tinylow.o semberia.s
ld65 -C c64-asm.cfg -S "$0801" tinylow.o -o tinylow.prg
```

To add a new command, it is first necessary to define a suitable name for it and calculate a hash for its recognition. The name consists of two bytes which are letters or BASIC tokens. The hash value is calculated according to the formula CV=LOW(first*2+second+(first>=$80)), or in assembly code:

```
LDA FIRSTLETTER
LSL
ADC SECONDLETTER
```

ASCII and token codes used to calculate the symbols are given in the following tables.

65/$41	A	74/$4A	J	83/$53	S
66/$42	B	75/$4B	K	84/$54	T
67/$43	C	76/$4C	L	85/$55	U
68/$44	D	77/$4D	M	86/$56	V
69/$45	E	78/$4E	N	87/$57	W
70/$46	F	79/$4F	O	88/$58	X
71/$47	G	80/$50	P	89/$59	Y
72/$48	H	81/$51	Q	90/$5A	Z
73/$49	I	82/$52	R		

128/$80	END	147/$93	LOAD	166/$A6	SPC(	185/$B9	POS
129/$81	FOR	148/$94	SAVE	167/$A7	THEN	186/$BA	SQR
130/$82	NEXT	149/$95	VERIFY	168/$A8	NOT	187/$BB	RND
131/$83	DATA	150/$96	DEF	169/$A9	STEP	188/$BC	LOG
132/$84	INPUT#	151/$97	POKE	170/$AA	+ (Addition)	189/$BD	EXP
133/$85	INPUT	152/$98	PRINT#	171/$AB	− (Subtraction)	190/$BE	COS
134/$86	DIM	153/$99	PRINT	172/$AC	* (Multiplication)	191/$BF	SIN
135/$87	READ	154/$9A	CONT	173/$AD	/ (Division)	192/$C0	TAN
136/$88	LET	155/$9B	LIST	174/$AE	↑ (Power)	193/$C1	ATN
137/$89	GOTO	156/$9C	CLR	175/$AF	AND	194/$C2	PEEK
138/$8A	RUN	157/$9D	CMD	176/$B0	OR	195/$C3	LEN
139/$8B	IF	158/$9E	SYS	177/$B1	> (greater-than operator)	196/$C4	STR$
140/$8C	RESTORE	159/$9F	OPEN	178/$B2	= (equals operator)	197/$C5	VAL
141/$8D	GOSUB	160/$A0	CLOSE	179/$B3	< (less-than operator)	198/$C6	ASC
142/$8E	RETURN	161/$A1	GET	180/$B4	SGN	199/$C7	CHR$
143/$8F	REM	162/$A2	NEW	181/$B5	INT	200/$C8	LEFT$
144/$90	STOP	163/$A3	TAB(	182/$B6	ABS	201/$C9	RIGHT$
145/$91	ON	164/$A4	TO	183/$B7	USR	202/$CA	MID$
146/$92	WAIT	165/$A5	FN	184/$B8	FRE	203/$CB	GO

A typical assembly code for implementing a new command, if it has no parameters, is

```
.ifdef CMDNAME
COMMAND:
    CMP #HASHVALUE
    BNE ENDCOMMAND
    ...  ; implement instruction
    JMP $A7E4 ; Continue interpreting with getting new character
ENDCOMMAND:
.endif
```

The program code of the commands should be positionally independent, so for branching within the implementation of the command use only relative jumps, and for memorizing intermediate values use unused locations in the C64 memory map.

So for example, if we want to add the command !DLIST (listing a directory without destroying the BASIC program) we see that it consists of the ASCII letter 'D' (value $44 which is less than $80) and the token 'LIST' (value $9B) , so that gives the hash value 2*$44+$9B+$0=$123. The low byte of this value is $23.

In the source file of SEMBERIA there is a comment where you should put your code.

```
;****** ADD NEW COMMANDS HERE
```

Atfer this comment put the following cote that implements !DLIST command:

```
.ifdef CMDDLIST
DLIST:
    CMP #$23      ; Hash of DLIST
    BNE ENDDLIST ; if not try other letter
    LDA #$01      ; filename length
    TAX
    LDY #$E8      ; there is a "$" at $E801 in ROM
    JSR $FFBD     ; set filename, A=namelen, XY=address of name
    LDA #$60
    STA $B9       ; set secondary address
    JSR $F3D5     ; send secondary address and filename
    JSR $F219     ; set default input device
    LDY #$04      ; skip 4 bytes (load address and link pointer)
FISIZE:
    JSR $EE13     ; input a byte from the serial bus
    DEY
    BNE FISIZE    ; loop
    LDA $90       ; STATUS variable
    BNE ENDDIR    ; check end of file
    JSR $EE13     ; read byte from the serial bus (block count low)
    TAX           ; keep it
    JSR $EE13     ; read byte (block count high)
    JSR $BDCD     ; print XA as unsigned 16 bit integer
    LDA #$20      ; Space character
    JSR $FFD2     ; print character to stdout
FINAME:
    JSR $EE13     ; read character
    JSR $FFD2     ; print character to stdout
    BNE FINAME    ; loop until zero
    JSR $AAD7     ; print carriage return character
    LDY #$02
    BNE FISIZE    ; skip 2 bytes next time (link pointer)
ENDDIR:
    JSR $F642     ; CLOSE
    JSR $F6F3     ; reset default input device
    JMP $A7E4     ; Continue interpreting getting new character
ENDDLIST:
.endif
```

Save the source and compile and link it using the commands

```
ca65 -D LOWVER -D CMDDLIST -mm near -t c64 -o custom.o semberia.s
ld65 -C c64-asm.cfg -S "$0801" custom.o -o custom.prg
```

If a command has parameters that are integer values in the range between 0 and 255, its assembly code has the following form

```
COMMAND:
    CMP #HASHVALUE
    BNE ENDCOMMAND
    JSR $B79B  ; get first byte parameter, result in X
    CPX #MAXPAR1 ; check parameter range
    BCS COMMANDERR
    ...
    JSR $B7F1  ; Check for comma then get second into X
    CPX #MAXPAR2 ; check parameter range
    BCS COMMANDERR
    ...
    JSR $B7F1  ; Check for comma then get third into X
    CPX #MAXPAR3 ; check parameter range
    BCS COMMANDERR

    ...  ; implement instruction
    JMP $A7AE  ; Main interpreter loop
COMMANDERR:
    jmp $B248          ; Illegal quantity error
ENDCOMMAND:
```

If the parameters cannot be represented by eight-bit values, e.g. the first two are integer sixteen-bit values and the third is a floating point number, a sample code might look like this:

```
COMMAND:
    CMP #CONTROLCODE
    BNE ENDCOMMAND
    jsr $0073          ; Scan for next symbol
    jsr $AD8A              ; Get Float parameter
    jsr $B7F7      ; Convert to 2 byte integer, Y=lo, A=HI
    ...      ; check parameter range
    BCS COMMANDERR
    ...
    jsr $AEFD          ; check for comma
    jsr $AD8A              ; Get Float parameter
    jsr $B7F7          ; Convert to 2 byte integer, Y=lo, A=HI
    ...      ; check parameter range
    BCS COMMANDERR
    ...
    jsr $AEFD          ; check for comma
    jsr $AD8A     ; Get Float parameter into locations $62-$66
    ...      ; check parameter range
    BCS COMMANDERR

    ... ; implement instruction
    JMP $A7AE  ; Main interpreter loop
COMMANDERR:
    jmp $B248          ; Illegal quantity error
ENDCOMMAND:
```

As an example of a new command with a parameter, we will add !CGOTO n, a version of the GOTO command that instead of a fixed line number allows a jump to a line calculated by an arithmetic expression. The assembly code for this command is:

```
.ifdef CMDCGOTO
cgoto:                  ; CGOTO line (calculated GOTO)
    cmp #$0F            ; Hash of CGOTO low (2*$43+$89+0)
    bne endcgoto        ; if not try other letter
    jsr $0073           ; Scan for next symbol
    jsr $AD8A           ; Get Float parameter
    jsr $B7F7           ; Convert to 2 byte integer, Y=lo, A=HI
    sty $14             ; Area for GOTO command
    sta $15             ; Area for GOTO command
    jsr $A8A3           ; Execute GOTO
    jmp $A7AE           ; jump to main interpreter loop
endcgoto:
.endif
```

Include it the same way as the command !DLIST and compile.

D64, TAP, WAV files and transfer to real C64

The assembler generates programs in PRG format and many C64 emulators support it. However, it contains only one single program,so it is often more practical to copy the PRG files into D64 (diskette emulation) files, TAP (casette tape emulation) files and WAV (sound based tape emulation) files, because they support multiple files. Although the defaulf version of SEMBERIA BASIC is already packed in these three formats, when building your own customized version of SEMBERIA BASIC, it is likely that you want to make them as well.

Emulator VICE comes with a tool called c1541, which is useful to create D64 files and to store PRG files into D64 files. If c1541 executable is available in your PATH, you can create the disk emulation file called semberia.d64 and put the prg files into using the folowing commands.

```
c1541 -format "semberia,1a" d64 semberia.d64
c1541 -attach semberia.d64 -write high.prg high
c1541 -attach semberia.d64 -write low.prg low
C1541 -attach semberia.d64 -write install.prg install
```

Other tool PRG-WAV (available at https://wav-prg.sourceforge.io/wavprg.html) is useful to make TAP and WAV files from PRG. Install it, find the required DLLs, and make it available in your PATH. To create TAP file that contains all three PRG files use the command

```
prg2wav --use-filename -s -t semberia.tap low.prg high.prg install.prg
```

To create WAV file that contains all three PRG files use the command (note that some sound cards do not require -i parameter):

```
prg2wav --use-filename -i -s -w semberia.wav low.prg high.prg install.prg
```

WAV files are huge, so they are not a good way to store tape content. They are intended to transfer SEMBERIA BASIC to the real C64. To do it, you need two tape recoders (one ordinary audio tape recorder and one Commodore 1530 datasette), and one cassette tape. Connect the ordinary tape recorder to your sound card. Hookup the lead from the LINE OUT jack on the sound card to the LINE or AUX IN on your tape recorder. Make sure that the mixer settings for the sound card have the LINE OUT enabled and adjusted to a level suitable for the tape recorder. Depending upon the model, the input on the deck might be a stereo jack, or might be two separate RCA sockets, so you may need to use an adapter or change the plugs on the end of the lead. Play the WAV file using the appropriate software like Audacity and press Record+Play on the tape recorder. After the recording is finished, connect 1531 datasette to your C64, insert the just recorded cassete, rewind it and try LOAD commands.

Customizing INSTALL version in DATA lines

If you want to make changes to SEMBERIA BASIC without using the source code, AS65 assembler and PRG format conversion tool, the INSTALL version can be used as an alternative. This is a more difficult way, but it is usable on the real C64.

Run the following commands to load the INSTALL version.

`LOAD "INSTALL",8`	for disk version
`LOAD "INSTALL",1`	for the tape version

The subroutine starting at line 60000 inserts machine code, starting at the address specified in variable SS (default 49152, which is set in line 1000). The last address of the machine code is in variable AA.

Lines 1020-1070 save the entered code to disk under the name EXBAS.

After executing with the command

```
RUN
```

You will receive (possibly a shortened or extended) version of BASIC extension located at the desired address. If you have not changed lines 1020-1070 it will also be recorded on diskette for later loading with

```
LOAD "EXBAS",8,1
```

If you want it to be recorded to tape, replace the line

```
1030 SYS 57812("EXBAS"),8,1 :POKE 193,SL:POKE 194,SH
```

with

```
1030 SYS 57812("EXBAS"),1,1:POKE 193,SL:POKE 194,SH
```

In this version it is possible to modify the BASIC extension, by removing the lines from the REM command describing the command you do not need with DATA lines after it until the next REM command (attention: if you remove the PPOS command, the LI command will not work, while the PPOS can work without LI).

It is also possible to add new commands with additional DATA lines. To do it, first convert the assembly code of the new commands into the machine code and write the decimal values. You can do it using native assembler on C64 (for example Profi Ass 64 2.0), cross assembler (like AS65) or even using pen, paper and 6502 instruction set table. For the previously written command !DLIST we get the following machine code in decimal:

```
201,35,208,69,169,1,170,160,232,32,189,255,169,96,133,185,
32,213,243,32,25,242,160,4,32,19,238,136,
208,250,165,144,208,30,32,19,238,170,32,19,238,
32,205,189,169,32,32,210,255,32,19,238,32,210,255,
208,248,32,215,170,160,2,208,216,32,66,246,32,243,246,
76,228,167
```

These values will be written in the DATA lines of the INSTALL version. The sum of these values is 10138. Add 1000 to this sum, and append the result 11138 to the last DATA line describing the command-

Now load the INSTALL version with

```
LOAD "INSTALL",8
```
for disk version

```
LOAD "INSTALL",1
```
for the tape version

Add the following lines to the installer

```
63200 REM !DLIST
63202 DATA 201,35,208,69,169,1,170,160,232
63204 DATA 32,189,255,169,96,133,185
63206 DATA 32,213,243,32,25,242,160,4
63208 DATA 32,19,238,136,208,250,165,144
63210 DATA 208,30,32,19,238,170,32,19,238
63212 DATA 32,205,189,169,32,32,210,255
63214 DATA 32,19,238,32,210,255
63216 DATA 208,248,32,215,170,160,2,208
63218 DATA 216,32,66,246,32,243,246
63220 DATA 76,228,167,11138
```

Insert a blank floppy disk and run the installer with

```
RUN
```

A new version of EXBAS will be saved soon and it will include a new command. Reset the computer. Test the new command using:

```
LOAD "EXBAS",8,1
NEW
SYS49152
!DLIST
```

The other example command !CGOTO is coded as

```
201,15,208,19,32,115,0,32,138,173,32,247,183,132,201,33,21,32,163,
168,76,164,167
```

The sum of these values increased by 1000 is 3552. Now load the INSTALL version as described above, add the following lines to the installer and run it.

```
63300 REM !cgoto
63302 DATA 201,15,208,19,32,115,0,32
63304 DATA 138,173,32,247,183,132,201,33
63306 DATA 21,32,163,168,76,164,167,3552
```

Type-in programs

A type-in program or type-in listing is computer source code printed in a home computer magazine or book. It was meant to be entered via the keyboard by the reader and then saved to cassette tape or floppy disk. The result was a usable game, utility, or application program. Today, they are more commonly published on web pages as programming tutorials.

By shortening the INSTALL program and then adding a part of the program that uses remaining commands, it is possible to write applications suitable for type-in: BASIC listing that we just type and run with RUN. If you are writing a BASIC program for type-in magazines and want your program to be typed together with the BASIC extension, delete lines 1020-1070 from the INSTALL program and replace them with the line

```
1020 SYS SS
```

Delete the DATA lines that define commands you do not use in your program and write the rest of the BASIC program that uses the remaining new commands.

For example, if our program uses only the !AT command, and we want to publish it as a Type-in program intended for those users who do not have the full SEMBERIA, we can publish program like this:

```
1000 SS=49152
1010 GOSUB 60000
1020 SYS SS
2000 PRINT CHR$(147)
2010 !AT 2,10: PRINT "WELCOME"
2020 !AT 2,15: PRINT "TO MY PROGRAM"
5000 END
60000 AA=SS
60010 SU=1000
60020 READ BB
60030 IF BB=1000 THEN RETURN
60040 IF BB<256 THEN GOTO 60070
60050 IF BB<>SU THEN PRINT "error near data line ";
PEEK(63)+256*PEEK(64)
60060 GOTO 60010
60070 POKE AA,BB
60080 SU=SU+BB
60090 AA=AA+1
60100 GOTO 60020
60110 REM start part
60115 DATA 165,20,24,105,16,141,8,3
60120 DATA 165,21,105,0,141,9,3,96
60125 DATA 32,115,0,201,33,240,3,76
60130 DATA 231,167,160,0,230,122,208,2
60135 DATA 230,123,177,122,10,230,122,208
60140 DATA 2,230,123,113,122,5654
60300 REM !at
60305 DATA 201,214,208,28,32,155,183,134
60310 DATA 2,32,241,183,164,2,192,40
60315 DATA 176,11,224,25,176,7,24,32
60320 DATA 240,255,76,174,167,76,72,178,4924
63200 REM final part
63205 DATA 76,8
63210 DATA 175,1259
63220 DATA 1000
63230 END
```

Merge HIGH or INSTALLed version with BASIC program

The LOW version is integrated with your BASIC program that uses SEMBERIA extensions, and it is quire easy to LOAD and RUN your BASIC program. However, it can be customized by reducing/adding commands only from source code using PC.

HIGH or EXBAS versions, however, require additional commands to be typed before loading and starting your BASIC progrm. If you have written the entire program using HIGH version, or custom version EXBAS made using INSTALL program, you may wish to integrate it with the BASIC program, because it is easier to start and distribute your program. It is possible to place machine code at the end of the BASIC program, between the last command and the variables. This is also the most popular place to host C64 machine code programs. Changes to the BASIC program will not be possible after this (unlike the LOW version), but saving and loading programs along with command expansion is done with the usual SAVE and LOAD commands. The first line that initializes the BASIC extension should be added to the program that uses SEMBERIA commands. By default it is SYS 49152, so the example looks like this:

```
0 SYS 49152
10 !CLRS
20 !AT10,5:PRINT "I LOVE YOU"
```

We run the program and test it in detail. When we want to write it so that the BASIC-extension machine code is placed after the BASIC program, we first look at where the end of the program is.

```
PRINT PEEK (45)+256*PEEK(47)
```

Let's write that value down. Let's say for example the returned value is 5092. Now change the first line to start the initialization from that address, being careful not to change the length of the BASIC program, so line 0 would look like this:

```
 0 SYS 05092
```

The new version of this line must have the same number of characters as the old version, so SYS 49152 has been changed to SYS 05092 and not SYS 5092. Now we will move the area where the BASIC variables start, increasing the BASIC program by the length of the code (eg 1970), immediately delete the variables, move the code to the given place and save the complete program.

```
!DPOKE 45,5092+1970
CLR
!MPOKE 49152,5092,1970
SAVE "APP",8
```

(If you have custom version without !DPOKE and !MPOKE, use the equivalent POKE and FOR-NEXT commands.)

Reset the computer and to start your BASIC program simply do

```
LOAD "APP",8
RUN
```