GIGA-BASIC

GIGASOFT

By André Claaßen and Gerd-Uwe Neukamp
Published by ABC Elektronic

CONTENTS

DISCLAIMER

The program has been developed and reproduced carefully, neither the author nor the distributors, however, can guarantee that the program and this guide are free of errors. Therefore GIGA-SOFT under no circumstances will be liable for any direct, indirect, Incidental or consequential loss of use, stored data, profit or contracts which may arise from any error, defect or failure of this software.

GIGA-SOFT has a policy of constant development and improvement of their products. We reserve the right to change manuals and software at any time and without notice.

INSTRUCTIONS FOR THE BASIC EXTENSION "GIGA-BASIC VERSION 1.00"

INTRODUCTION:

Although the Sinclair QL comes with a real good BASIC, some commands are missing, which would offer the full power of the QL. This extension set should increase your motivation to program in BASIC. With over 70 commands and functions, GIGA-BASIC is a useful extension for the QL. Before starting work with GIGA-BASIC you should read this manual carefully. You should never work with your original copy of GIGA-BASIC. To obtain working copies a backup program is included. To start this program enter 'exec_w mdv1_clone'. You can backup GIGA-BASIC up to three times.

The extension includes the following groups

- graphics
- sprite handling and sprite animation
- base conversion
- full screen BASIC editor
- direct access to medium
- multitasking clocks
- mouse driven screen oriented menus
- pull-down-menus
- multitasking control commands
- programmable function keys
- others

GRAPHICS

- **PAINT** #dev, x, y
  Fills an irregularly bordered area of the chosen screen with the ink colour.

dev: device number of the screen
x: x= coordinate
Y: y= coordinate

DIRECT ACCESS TO MEDIUM

- **GET** #dev, variable (,variable)
  Gets a value from medium and writes it into the variable. The type of the value depends on the type of the variable.
  Example: GET #4, integer%

  #dev: device number
  variable: any type and number of variables

- **BGET** #dev, byte (,byte)
  Gets byte from medium and puts it into the variable.

  #dev: device number
  byte: any type and number of variables

- **PUT** #dev, variable (,variable)
  Writes value to Microdrive. Any variable type is allowed.

  #dev: device number

  variable: any type and number of variables.

- **BPUT** #dev, byte (,byte)
  Writes byte to Microdrive.

  #dev: device number
  byte: variable which gets a byte

- **SET_POINTER** #dev, pointer
  Sets file-pointer to new position. With this command it is possible to have direct access to Microdrive (or FLP, HDK, FDK and so on).

  #dev: device number
  pointer: longword containing the pointer

- **GET_POINTER** (#dev)
  Gets the pointer of the selected Microdrive.
  Example: pointer = GET_POINTER (#dev)

  #dev: device number
  pointer: variable containing the pointer

BASE CONVERSION

The following functions provide an easy way to convert bases.

hexnum$ = **CHEX$**(decimal)

Converts a decimal value into a hexadecimal string
hexnum$: string which will contain the hex number
decimal: variable containing the decimal number

decimal = **CHEX**(hexnum$)

Converts a hexadecimal string (max. 32 bit) into a decimal number.

decimal: variable which will contain the decimal number
hexnum$: string containing the hex number

binary$ = **CBIN$**(decimal)

Converts a decimal number into a binary string (32 bit).

decimal: variable which will contain the decimal number
binary$: string containing the binary number

decimal = **CBIN**(binary$)
Converts a binary string into a decimal number.
Decimal: variable which will contain the decimal number
binary$: string containing the binary number

MULTITASKING CONTROL COMMANDS

The following commands are intended to control the multitasking capabilities of the 04. Now it is possible to delete, suspend or activate jobs from BASIC.

JOB_INF #dev

This command display, a list of all active Jobs. A job is a program working in the background. Additionally you can see the priority, the owner job, the base address and the tag number. Job 0 is the BASIC interpreter. For further information on multitasking refer to the Sinclair User Guide.

#dev: device number

SUS_JOB jobnr, tagnr, timeout

Suspends a job for a period.

jobnr: jobnumber
tagnr: tagnumber
timeout: Number of frames the job being deactive (-1: infinite).

REL_JOB jobnr, tagnr

Releases a suspended job. This command is the reverse of SUS_JOB.

jobnr: jobnumber
tagnr: tagnumber

PRIOR_JOB jobnr, tagnr, priority

Sets the priority of a job. Priorities are allowed in the range from 0 to 127, where 127 is the highest priority. If the priority is high, more time is available for the job.

jobnr: jobnumber
tagnr: tagnumber
priority: priority

SPRITES AND ANIMATION

GIGA-BASIC offers a great number of efficient commands for development and animation of sprites. So it is easy to generate action games or programs using icons. Sprites are organised in a 32 x 20 matrix and are flicker-free

Important definitions:

Sprite-data-block (sprdat): This is a memory block which contains the bytes for the shape (mask) of the sprite. A sprite shape contains 160 bytes. Every Sprite-data-block can be attached to every sprite.

Sprite-number (sprnr): A sprite will be activated under a sprite-number. Under this number the sprite can be moved over the whole screen.

- **SPRDIM** spritenr, datanr, animtenr

Reserve memory for sprites. The defaults are:
SPRDIM 4,16,4

spritenr: number of possible sprites
datanr: number of possible Sprite-data-blocks

animtenr: number of the sprites which can be animated

- **SPRCLR**

  SPRCLR releases the memory allocated by SPRDIM. All defined Sprites are lost.

INVMASK #dev, x, y, sprdat

Prints a sprite mask onto the screen. The coordinates are relative to the left upper edge of the selected window. The coordinates have pixel size 4. (This is not a sprite, only a mask will be drawn).

dev: device number
sprdat: sprite-data-block

- **SPRON** sprnr, sprdat

  Activates a sprite with a sprite-data-block.
  Note: This command does not have any effect on the screen. The sprite will not be visible until it is activated by the MOVESPR command

sprnr: sprite number
sprdat: sprite-data-block

- **SPROFF** sprnr

  Removes the selected sprite.

sprnr: sprite number

- **REFRESH**

  Important after "CLS'. All active sprites are refreshed.

- **INVSPRITE** sprnr
  The chosen sprite is inverted.

sprnr: sprite number

MOVESPR sprnr, x, y (,sprdat)

Sets a sprite to a new position. The optional sprite-parameter is intended to change the appearance of the sprite. If no sprite-data-block parameter is given, the sprite image does not change.

sprnr: sprite number
x, y: *absolute pixel coordinates
sprdat: sprite-data-block

Several sprites can be moved using only one command. This type of motion is named animation. It is a really easy task to move rockets, men, cars and other things now.

- **SETANIMATE** sprnr, sprdat(,sprdat1)(,sprdat2)
  This command has as many parameters as you want to. The given sprite-data-block are connected in series.

  Note: Before using ANIMATE, you have to initialise the routine with the SETANIMATE command. A maximum of 16 sprite-data-blocks may be connected.

  sprnr: sprite number
  sprdat: sprite-data-block

- **CLRANIMATE** sprnr

  The selected sprite entry will not be animated after the use of CLRANIMATE.

  sprnr: sprite number

- **STEPSPRITE** sprnr, xstep, ystep, statx, staty

  This command CIO be used after every SETANIMATE. You can change the direction and speed of the animation in your basic program.

  sprnr: sprite number
  xstep: stepsize x
  ystep: stepsixe y
  statx: 0 After reaching the border of the screen inverts the x-direction.
  1 After reaching the border appear at the other side.
  2 After reaching the border kill the sprite.
  staty: Same as statx but referring to the y-direction.

- **ANIMATE**
  Moves all sprites which are declared with the SETANIMATE command over the screen.

  Sprite = **COLLISION**(sprnr)

  Asks whether two sprites are overlaid. If it is true COLLISION returns the sprite number, otherwise -1.

  sprnr: sprite number
  sprite: If the sprite isn't in contact with another sprite -1 will be returned, otherwise the sprite number.

SPRITE DEFINITION COMMANDS

Sprites can be defined for MODE 4 or MODE 8. The following is an example of an eight colour sprite:

```
100 :
110 SPRDEFBLOCK starship
120 :
130 SD8 "................"
140 SD8 "................"
150 SD8 ".......11......."
160 SD8 "......1111......"
170 SD8 "....22222222...."
180 SD8 "..333333333333.."
190 SD8 "...3333333333..."
200 SD8 ".....7....7....."
210 SD8 "....7......7...."
220 SD8 "...7........7..."
```

The four colour example:

```
100 :
110 SPRDEFBLOCK disk
120 :
130 SD4 "................................"
140 SD4 ".################################."
150 SD4 ".##############################..."
160 SD4 ".################################."
170 SD4 ".################################."
180 SD4 ".##############....##############."
190 SD4 ".############........############."
200 SD4 ".##########..........##########."
210 SD4 ".############........############."
220 SD4 ".##############....##############."
230 SD4 ".################################."
240 SD4 ".###############..###############."
250 SD4 ".###############..###############."
260 SD4 ".################################."
270 SD4 "................................"
```

The colours are set in the following forms

MODE 4

```
Red         : '1'
green       : '2'
white       : '3','#'
black       :all other characters
```

MODE 8

```
blue        : '1'
red         : '2'
magenta     : '3'
green       : '4'
```

```
cyan        : '5'
yellow      : '6'
white       : '7','#'
black       : all other characters
```

- **SPRDEFBLOCK** sprdat

  Clears the selected sprite-data-block and prepares it for a new definition.

  sprdat: sprite-data-block

  SD4 *defblock$*

  Command to define a four colour sprite. Up to 20 commands can be used after a SPRDEFBLOCK command. The string must be 32 characters long.

  SD8 *defblock$*

  Command to define an eight colour sprite. Up to 20 commands can be used after a SPRDEFBLOCK command. The string must have a length of 16 characters.

  SPRLOAD *name$*

  With this command you can load previously defined sprite-data-block. Before you use this command enough space must be reserved by SPRDIM.

  Example: PRLOAD *MDV1_PACMAN_SPR*

  SPRSAVE *name$*

  If you want to save the allocated sprite area you can use this command. Only the area for sprite-data-block will be saved.

  flag = ***SPRACTIVE*** *(sprnr)*

  With SPRACTIVE you can ask whether a sprite in active. 1 is true and 0 is false.

  sprnr: sprite number

  *x=****SPRXPOS****(sprar)*

  *y=****SPRYPOS****(sprnr)*

  With these functions you can find out the location of a sprite.

  sprnr: sprite number

MENU CONTROL COMMANDS

The following commands support your friendly screen orientated menus. Now you can program mouse-driven menus as with the APPLE MACINTOSH™

The handling is very simple. With commands like MENUPR or MENUBLOCK you define a BLOCK. This block can be manually inverted or selected with the MOUSE function. Possible input media are the cursor keys or a mouse with the *ABC-interface* (included in the big ABC package).

Example :

```
100 :
110 REMark small example menu
120 :
130 CLS
140 PRINT "M E N U"
150 PRINT: PRINT
160 MENUPR 1," Start a program"
170 MENUPR 2," List a program*
180 MENUPR 3," End"
190 :
200 a=MOUSE
210 :
220  SELect on a
230     =1: Start
240     =2: LIST
250     =3: STOP
260  END SELect
```

After entering and starting the program, the menu points appear as if they were printed with the PRINT command. An arrow appears, too. This arrow can he moved over the whole screen. If the arrow is in range of a menu point this will be inverted. So you can see exactly what you have chosen. By pressing the button or space the selected menu number will be returned.

- **SETMDEV** mode

  Selects input medium for the menu commands.

  mode: 0: keyboard (cursor keys / space)
  1: mouse

- **MENUDIM** number

  Reserves memory for the menu points. Space for pull-down-menus will be automatically allocated.

  number: the maximum number of menu points

- **MENUBLOCK** #dev, b1knr, x, y, xO, yO

  This command marks a block with the chosen menu-block-number.

  #dev: device number of a screen
  blknr: menu-block-number
  x, y: size of the block
  xO, yO: position relative to the selected window (offset)

- **MENUPR** #dev, blknr, text$

  Prints a text an the screen similar to the print command and activates it as a menu block.

  #dev: device number of a screen
  blknr: menu-block-number
  text$: text
  The separator ';' is allowed.

- **ICON** #dev, blknr, sprdat, x, y

  Similar to the INVMASK command it displays a sprite block on the screen. The difference is that ICON marks it menu block. With this command it is possible to access symbols in a similar way as the MENUPR command. You can define ICONs and use them for defining MACINTOSH™ style programs.

  #dev: device number of a screen
  blknr: menu-block-number
  sprdat: sprite-data-block
  x, y: pixel coordinates relative to the window

- **INVBLOCK** b1knr

  Inverts a block.

  blknr: menu-block-number

- **CLRBLOCK** b1knr

  Clears a block.

  blknr: menu-block-number

- nr=**MOUSE**(x, y)

  Displays an arrow which can be moved over the screen by using the mouse. With the arrow you can select an item.

nr: if no menu point is chosen, -1 will be returned, otherwise the otherwise the menu-block-number will be returned.
x, y: start coordinates of the arrow

- x = MXPOS, y = MYPOS

  These functions return the position of the arrow after pressing the SPACE-key.

PULL-DOWN-MENUS

This a new type of menu technique. On top of the screen you can see a headline holding the menu points. If you move the arrow to one of the points, a window will be opened with a submenu. Now you can choose the point you want in the submenu. With the Pull-Down-Menus you can handle a great number of menu points on a very small room.

Example:

```
100 SPRDIM  :REMark Reserves space for the arrow
110 MENUDIM :REMark Allocates space for the pull-down-menu
120 :
130 MENU 0,0,1, "Addresses"
140 MENU 1,0,1, "Clear"
150 MENU 2,0,1, "Input"
160 MENU 3,0,11 "Edit"
170 :
180 MENU 0,11,1, "File"
190 MENU 1,1,1, "Load"
200 MENU 2,1,1, "Save"
210 :
220 MENU 0,2,1, "Exit"
230 MENU 1.2,t, "Reset"
240 MENU 2,7,1, "Basic"
250 :
260 SETMENU :REMark Clears the screen and shows the menu headline
270 :
280 GETMENU :REMark Shows the arrow and gets the menu point
290 x = HMENU
300 y = VMENU
310 :
```

- **MENU** vnr, hnr, active, string$

  Command to define a pull-down-menu.

vnr: Vertical coordinate. The headline has the coordinate zero.
Note: The menu points within the headline (vnr=0) must be defined In ascending order. Every headline point must

have a submenu. A maximum of 10 Items can be defined in the vertical direction.

hnr: Horizontal coordinate. The number of horizontal items is restricted to a maximum of 8. The total length of the items in the headline must be selected to fit according to the selected screen mode. This is important for compatibility between mode 8(256) and mode 4(512).

active: Flag which selects whether you can access the menu point or not

string$: Text of the menu point. The length is restricted to 14 characters.

- **SETMEMU** paper1, paper2, actcol, pascol

  Clears the whole screen. Displays the headline.

  paper1: is the screen colour
  paper2: border colour of the headline
  actcol: Colour of the active menu points
  pascol: Colour of the passive menu points

- **GETMENU** x, y

  Displays the arrow and allows the user to select menu points.

  x, y: start position of the arrow

  Default: GETMENU 256,100

- **ACT1VE** vnr, hnr, active

  Activates and deactivates menu points.

  vnr: Vertical coordinate of the menu point
  hnr: Horizontal coordinate of the menu point
  active: Flag, 1-active, 0-inactive

x = **HMENU**
y = **VMENU**

With these functions you can get the position of the chosen menu point.

Possible range:
HMENU (0-7)
VMENU (1-9)

PROGRAMMABLE FUNCTION KEYS

Directly after starting the BASIC extension, the function keys are programmed, information about the assignments can be gained by pressing "F1". This assignment can easily be changed by the user. Furthermore the function keys can be switched off if they would disturb the function of other programs.

- **KEYS** #dev

  Lists all function key assignments to the specified device.

  #dev: device number (default is 1)

- **KEY** keynr, string$

  Allows the user to change the function key assignment.

  keynr: Number of the function key (1 to 10), numbers greater than 5 are activated by pressing the shift key simultaneously.
  String$: String containing the command (max. 32 characters). Example: KEY 1, 'LIST'&CHR$(10)

- **KEYSON**
  Turns function keys on.

- **KEYSOFF**
  Turns function keys off.

CLOCK COMMANDS

It is possible to display either a digital or an analogue-clock on the screen. There is also the possibility of changing colour and size to adapt the clocks to own programs.

- **DCLOCK** on, x, y, paper, ink1, ink2
  Displays a digital clock.
  Default DCLOCK 1, 340, 0, 2, 7, 4
  on: flag, 0-removes the clock, others activate the clock
  x, y: right upper coordinate of the clock in pixel coordinates
  paper: paper colour
  ink1: ink colour
  ink2: border colour

- **ACLOCK** on, x, y, size, paper, ink1, ink2, ink3, ink4

  Displays an analogue- clock.
  Default ACLOCK 1, 0, 0, 40, 0, 2, 2, 4, 6

on:	flag, 0-removes the clock, others activate the clock
x, y:	right upper coordinate of the clock in pixel coordinates
size:	vertical size of the clock
paper:	paper colour
ink 1-4:	ink colour for the hands of the clock and the circle around it

OTHER COMMANDS

- **CAT** #mdvnr

  Displays the directory of the specified drive in a formatted form. Furthermore it displays the number of blocks (512 bytes) each program uses.

  #mdvnr: number of drive (default is 1)

- **DUMP** #dev

  Displays all variables with contents, procedures and functions with line numbers.

  #dev: output device (default is 1)

- **COMMANDS** #dev

  Lists all new BASIC commands with their start address on the output device.

  #dev: output device (default is 1)

- **HRDCOPY** inv

  Prints hardcopy on EPSOM-compatible printers. Through technical restrictions, it is only possible to print a maximum of 480 horizontal points.

  inv: flag, 1-inverted print, 0-normal print

- **SYSTEM** #dev

  Displays the system variables on screen.

  #dev: output device (default is 1)

- a = FREE

  Returns the amount of free BASIC memory.

- **SCREEN** #dev, linenr, tab

  Default: SCREEN 11, 1, 3
  This command enters the screen editor. It allows the user to edit BASIC programs in a way similar to QUILL. Unlike a normal ASCII-Editor all entered lines are syntactically checked by the interpreter.

Note: The interpreter will not accept lines after a program break if the functions and procedures are not reinitialized. This is possible by using the CLEAR command, which will produce the message 'PROC/DEF CLEARED'. After this message the work with the screen editor can go on.

#dev: window number to edit in
Linenr: line number which will be displayed
Tab first step size of the inbuilt tabulator

The editor accepts the following key sequences:

Cursor Up	
Cursor Down	
Cursor Right	
Cursor Left	
ESC	Leaves the editor
TABULATE	Tabulator
SHIFT&ALT&UP	Jumps to start of program
SHIPT&ALT&DOwN	Jumps to end of program
ALT&UP	Page up
ALT&DOWN	Page down
CTRL&RIGHT	Deletes character under the cursor
CTRL&LEFT	Deletes character at the left of the cursor
CTRL&ALT&LEFT	Clears basic line
CTRL&ALT&RIGHT	Deletes all characters at the right of the cursor
SHIFT&UP	Jumps to the first line of the screen
SHIFT&DOWN	Jumps to the last line of the screen
ALT&LEFT	Jumps to start of line
ALT&RIGHT	Jumps to end of line

- **SETFONT** #dev, fount1, fount2

Gives the user the possibility of using a self-defined Character Set. It is possible to define up to two character sets at one time, in which case a character is displayed from the first character set, If defined there, if not defined, it is taken from the second and, if it also is not defined there, the first defined character of the second set is displayed. To select the inbuilt fonts of the QL, just enter 0(zero) for the start address of the font.

fount1: start address of the first font
fount2: start address of the second font+

Example: (Using the character set 'BIG_CST')

```
100 a = RESPR (1024)             :REMark Reserve space for font
110 LBYTES 'mdvl_BIG_CST',a      :REMark Load new font
120 FOR channel = 0 TO 2         :REMark Loop
```

```
130 SETFONT #channel              :REMark Activate new for font
140 CLS #channel                  :REMark for every window
150 END FOR channel               :REMark End loop
```

- **MONSCR** mode

  Activates the switch-on-status of the windows for the monitor mode

  mode: selects 4 or 8 colour mode

- **TVSCR** mode

  Activates the switch-on-status of the windows for the television mode

  mode: selects 4 or 8 colour mode

- **SETMON** #dev, xsize, ysize, x0, y0, paper, strip, ink, borderwidth, bordercolour

  Changes the default window in monitor mode

- **SETTV** #dev, xsize, ysize, x0, y0, paper, strip, ink, borderwidth, bordercolour

  Changes the default window in television mode

- Mode = **SETMODE**

  Returns the screen mode
  4 - four colour, 8 - eight colour.

WINDOW COMMANDS

The window commands allow the user to work with the real windows. With these commands it is possible to save the background of a window before writing to it and to restore this background after closing the window. This technique is known as 'refreshing'.

- **SCRSTORE** nr, xs, ys, x, y

  Saves and area of the screen

  nr: a number from 0 - 15. This number represents the label for the saved screen. It has to be specified in the other commands referring to the saved screen area.
  xs, ys: size of the window to be saved
  x, y: left upper position of the window to be saved

- **SCRLOAD** nr

  Redisplays a saved area of the screen

  nr: label number (0 - 15)

- **SCRCLEAR** nr

  Clears the part of memory containing the saved screen

  nr: label number (0 - 15)

INDEX