

POINTER'S TOOLKIT

WRITER'S TOOLKIT

A SOFTWARE PRODUCT FROM
JOCHEN MERZ

ABC Electronic

POINTER'S TOOLKIT & WRITER'S TOOLKIT

(C) 1987 Jochen Merz

You may use both toolkits together as well as separately. If you wish to load one of them only, please have a look at the instructions of that one.

In the following manual square brackets [] mean options (you must not type in this parameters, you may). Text printed *italic* shows examples.

Load both programs as usual after the startup or react-boot or by typing

LMN advl_BOOT or *LMN flpl_BOOT*

Please have a look into both sections of this manual. There is also an example program supplied on your disc/cartridge. After having loaded both toolkits, run it by typing

LMN advl_DMO or *LMN flpl_DMO*

QL, QXOS are registered trademarks of Sinclair Research Ltd.

Copyright of the code of Pointer's Toolkit & Writer's Toolkit and this manual by Jochen Merz, 1987.

No part of this software or documentation may be reproduced in any form. Unauthorised copying, hiring, lending or sale and respurchase is prohibited.

In no circumstances will either the author or the distributor be liable for any direct, indirect or consequential damage or loss including but not limited to loss of use, stored data, profit or contracts which may arise from any error, defect or failure of this software.

This program contains a coded number. It is possible to find out the initiator of illegal copies.

POINTER'S TOOLKIT

(C) 1987 Jochen Merk

This toolkit is not intended to be a rival to SuperToolkit II or other toolkits but to be a complementary toolkit to SuperToolkit II and/or QWM. It contains commands which you cannot find in SuperToolkit II, for example commands supporting the pointer interface of QWM or QRAM itself. Before loading Pointer's Toolkit you should have invoked SuperToolkit II (if present) by typing `TK2_EXT` (if necessary). After loading of Pointer's Toolkit you should not use the `TK2_EXT` command as it over-defines some of pointer's Toolkit's new commands (e.g. `EXTRAS`, `RESIR`).

To load Pointer's Toolkit type in

`LMN adv1_BOOT_ITK` or `LMN fapl_BOOT_FTK`

When loaded you get a message about the state of the pointer interface. When there is no pointer interface (or not initialised till now) you can read

No Pointer Interface

If you used the command `POINTER` (if you own a Sandy SuperQboard with mouse) or loaded `PTR_EWD` or `PTR_LMI`, you get

Pointer interface V1.06 (perhaps with a different version number)

If there is also a window manager (necessary to use QRAM) then it is also displayed:

Window Manager V1.05

Pointer's Toolkit is ready now.

Pointer-Commands**PTINFO [window]**

shows information about the current pointer interface and window manager (if there is any) to the selected window. If there is no window number specified, Channel #1 will be used. The messages are the same as the load-messages, subject to the current state.

PSET [window,]x,y[,wflag]
 sets the pointer to the position x,y. If there is no window number and no wflag then the pointer is set absolute to the screen coordinates (0,0 is the topmost left pixel of the screen). If there is a window number (default is #1) and wflag then the pointer will be set relatively to the top left corner of the specified window. PSET does not display the pointer! Works only with pointer interface!

PSET 50,50 sets the pointer to the absolute position 50,50.

PSET #2,30,30,1 sets the pointer to position 30,30, relativ to window #2.

PSET #2,30,30,1 sets the pointer to position 30,30, relativ to window #2.

PREAD [window,]END\$
 displays the pointer and waits until the event specified by END\$ occurs. You can move the pointer, press keys and/or mouse-buttons, even select other jobs by hitting their windows. The key- and button-events are only effective if the pointer is in the specified window. Default window is #1. Works only with pointer interface! You can give the following events for END\$:

S Return if SPACE or ENTER or a mouse-button is pressed. (Read the result with PKY\$).
 P Return if any key is pressed. (Read with PKY\$).
 U Return if no key is pressed (e.g. move with held button into a window and release it).
 M Return if pointer moves.
 O Return if pointer moves out of window (immediate return if pointer already out of window).
 I Return if pointer moves into window (immediate return if pointer already in window).

PREAD 'I' finish when pointer moves into window #1.

PREAD #2, 'P' finish when a key is pressed while the pointer is in window #2.

During the work with Pointer's Interface it was noticed that PREAD while reading other windows than #1 did not work properly. If you re-open channel 1 at the start of your program everything works fine. Perhaps this is due to the the version of the pointer interface. If you have any problems with PREAD try as your first program line:

1 OPEN #1,0\$

PKY

is a function which returns the code of the last keypress (there must be a PKEYAD before reading PKY). SPACE returns 1, ENTER 2. The left mouse-button also 1, the right 2.

PRINT PKY returns 27 if the last keypress, while the pointer was visible, was ESC.

There are two variables which return the position of the pointer:

PX
returns the x-position relativ to the window specified by the last PKEYAD command.

PY
returns the y-position relativ to the window specified by the last PKEYAD command.

Window Commands

To realise pulldown-windows or the like you have to save the area behind the area occupied by the pulldown-window. After closing the pulldown-window you can restore the original background. We recommend (as there is nothing to do wrong) to open a window, say #3, which covers the whole screen:

OPEN#3,CON_512x256a0x0

You can save this before pulling down another window by using

WSAVE#3

and after closing the pulldown-window

WICAD#3

The only disadvantage is: the whole window needs 32kBytes of memory, even if you modify just a little part of the window. You must not re-define the saved window!!!

WSAVE [window]

saves the contents of the specified window so that you can restore it at later date. If you do not specify the window, window #1 will be used. This command works only with the pointer interface being installed.

WSAVE #2 saves window #2.

WLOAD [#window][,1]

restores the specified window (default is #1) to the state you saved it last time. The memory occupied by the save area will be released if there is no ,1 parameter following the window number. If there is a parameter you can re-WLOAD the window at a later date again. This command works only with the pointer interface too.

WLOAD #2,1 restores the window saved in the last example, the save area remains.

WLOCK [#window]

unlocks a window, i.e. other jobs can't lock it by laying their windows above SuperBASIC's. The window number must be the number of the first opened window, #0 normally. This feature is implemented to make it possible to write a clock which should be compiled and run as an own job. If there is another job running which covers the clock's window totally or partially then the clock stops running. If you unlocked all windows of the clock, it will continue to run the same way as without pointer interface. Naturally this command works only with the pointer interface.

WESET type[,mode]

sets the windows #0, #1 and #2 to pre-defined size. There are seven pre-defined definitions you can specify for type, numbered from 0 to 6. You may specify a display mode optionally which will be selected if entered. Otherwise you stay in the current mode. All definitions redefines the window that there is about half of the screen free for windows of other jobs. If you use the normal monitor mode there is no way to select other jobs by pointing to their windows and select them (e.g. a QRAM window), as SuperBASIC covers them totally. If you use one of the new definitions normally there is at least a corner of other windows visible which you can select.

WESET 0,4 selects display mode 4 and gives you little monitor windows.

DMODE

is a function which returns the current display mode.

PRINT DMODE returns 4 for mode 4 and 8 for mode 8.

Jobs**CLOCK [x,y]**

gives a running digital clock. SuperToolkit's CLOCK does not work well with the pointer interface, you must select it with CTRL C or the mouse, but SuperBASIC stops. The new clock displays a little window at the top right corner or wherever you wish if you specify the coordinates. This clock will run even if other windows cover it. If you specify coordinates which are out of range the clock kills itself without reporting an error. Sometimes you have to press CTRL C to return to SuperBASIC but the clock will run continuously.

CLOCK 0,0 puts the clock at the top left corner.

FREE [x,y]

displays the current free memory in kilobytes (1024 Bytes). A job named Free is created which opens a window at the top of the screen, left beside the clock (if there is a default one). Of course you can give the coordinates of the position. This job behaves the same way as the clock does.

FREE creates a window at the top right of the screen.

BLANK [time]

creates a job named Blank which controls the keypresses of the computer. If there was no keypress within a specified time (in seconds) then the screen will be blanked until another keypress occurs. This should save your monitor if you wish. If you do not specify a time, 5 minutes are assumed.

BLANK 15:60 creates a job which blanks the screen after 15 minutes without keypress.

RJOB_A

kills all currently running jobs except SuperBASIC. This is necessary if you wish to load resident extension for example.

SUSJOB jobno,tag,time

suspends the job specified by jobno and tag the specified time (in 1/60 seconds). This is useful if you wish to disable the BLANK-job a given time or to deactivate the net-file-server for some reason.

SUSJOB 0,0,60 is the same as PAUSE 60 but you cannot abort the pause.

Memory Management**REMM(*bytes*)**

is the same function as the normal REMM but you never get the message 'not complete' if there are any jobs running. In this case the requested number of bytes will be taken from the common heap instead of the resident procedure area. You can use this memory for the same purpose as normally as this memory will never be released (until you reset the computer, of course!).

POKE\$ *addr*,*st\$*

puts the string *st\$* continuously from address *addr* upwards byte wise into the memory. The length of the string must not be 0.

PEEK\$(*addr*,*length*)

is the reverse function to the command POKE\$. *addr* is the starting address and *length* the number of bytes to be read from this address upwards and it is also the length of the string which will be returned from this function, *length* must not be 0.

s\$=PEEK\$(131072,32768) reads the contents of the whole screen into a variable *s\$*.

You can put this contents back with

POKE\$ 131072,*s\$*

Other

EXTRAS {channel}
lists all extra-commands and functions the same way SuperToolkit's EXTRA does, but uses the whole screen. You may pause the output by pressing CTRL F6 and interrupt the listing by pressing BREAK. Default channel is #1.

NETNO
is a function which returns the net-number of the own station. This may be useful if you use the net-file-server.

ED\$({window,'Text'})
is a function. The string 'Text' will be displayed in the specified window (default #1) and could be edited the usual way. ENTER, Cursor up or Cursor down finish the input. The edited string will be returned.
`a$=ED$(#2,'Night')` writes Night to window #2. Now you may edit Night. Delete twice left and enter oe, then press ENTER. a\$ contains now Nice.

Mathematical functions which are neither defined in SuperBASIC nor in SuperToolkit II:

SGN(x)
returns the sign of x: -1 if x is negative, 0 if x is 0, 1 if x is positive.

FRAC(x)
returns the unsigned value behind the decimal point of x.

PTK_EXT
re-defines all commands of Pointer's Toolkit. Commands like RRSPPR, which exist in SuperToolkit II also, could be re-defined to Pointer's Toolkit version if you defined them to SuperToolkits version by using TK2_EXT.

REIN device_filename,old\$,new\$

creates a job named **Rechange** which replaces in the file **device_filename** each **old\$** to **new\$**. **old\$** and **new\$** must have the same length (but not 0). Upper and lower case letter will not be distinguished. This command also uses the default-directory of **DATA_USE**, if it exists. You must never remove the job, it should kill itself.

REIN **mlv1_ROOT**,**'flp'**,**'mlv'** swaps each **flp** to **mlv** in the file **ROOT** on device **mlv1_** (F1P or Flp etc. will also be swapped!). If you own SuperToolkit II or a floppy disc controller which has **DATA_USE**, and **DATA_USE** is currently **mlv1_**, you could enter instead:

EXTG **ROOT**,**'flp'**,**'mlv'**

WRITER'S TOOLKIT

(C) 1987 Jochen Kern

To load Writer's Toolkit type in

LRN wkt1_WTK or **LRN fpl1_WTK**

You can use Writer's Toolkit without using the Pointer Interface or SuperToolkit II of course.

Writer's Toolkit writes only to the screen if display mode 4 is selected. This is because in mode 8 the resolution is too low; therefore the characters are too fat.

After loading Writer's Toolkit the following fonts are available:

Font #0: Antiqua12

Font #1: PGyptienne 20

Font #2: Antiqua20

Font #3: Shadysi 20

Font #4: Old English 24

Font #5: Universality Roman 24

Font #6: University Roman Italic 24

Font #7: Grotengro 24

Font #8: Helvetica Light 16

Font #9: Helvetica Bold 16

You can print out this list (and an extended one if you loaded other fonts) by issuing the new command

W_FONTS [channel]

writes this list to the specified channel (default is #1, as always). All defined fonts will be listed with their number, their name and the current font will be marked with a '>'. The number behind the font's name is the bit-height of the font.
W_FONTS #2 lists to channel #2.

The command

W_FONT fontnumber

selects a font. fontnumber should be in the range 0 to 9 if there are no other fonts loaded.

W_FONT 4 selects Old English. You may check it with **W_FONTS**.

W_INK colour

sets the ink for the WRITH command. There is no background or paper colour; this is the current background. Possible colours are 0, 2, 4, and 7.
W_INK 4 sets the ink to green.

WRITH x,y,text\$

prints the string `text$` in the current ink by using the current font at the absolute position `x,y`. Notice: WRITH writes only if the current display mode is mode 4. If `text$` is too long to fit into the screen then there will be no error message. There should be no control code within `text$`.

`W_FONT 2;W_INK 7;WRITH 0,0,'Hello'` prints white Hello by using font #2 at the top left corner of the screen.

W_WIDTH(char\$)

is a function which returns the width of the first character of `char$`. As all fonts are defined proportional it could be very useful for some application programs to know how wide a character is.

You may load other fonts of course (fortunately Writer's Toolkit fonts are compatible with Qwriter's). The command

W_LOAD device_fontname,fontnumber

loads the font `fontname` stored on device `device` (e.g. `f1p1_OldEnglish`) and defines it as font number `fontnumber` (should be in range 10 to 31, 0 to 9 may be over-defined of course). If the given filename does not exist, a `_fat` will be appended and another try will be made. If there is still no file and a `DATA_USG` default-directory exists, then this default will be placed before the filename and a final try will be made. You may write

`W_LOAD f1p1_OldEng.fnt,11` instead of

`W_LOAD f1p1_OldEng.fnt,11` if `DATA_USG` is `f1p1_` or

`W_LOAD f1p1_OldEng.fnt` if there is no `DATA_USG`.

W_OFT [pause[,beep]]

lets you specify a pause (in 1/60 seconds) between each character printing and a beep frequency which gives a 'click' when the character is being printed. `W_OFT` and no parameter is the default setting: no pause and no beep.

`W_OFT 10,100;WRITH 40,40,'An effect!' Just try it!`