EPROM MANAGER II

THING & EPROM UTILITY SOFTWARE

FOR THE QL

WRITTEN BY JOCHEN MERZ

EPROM MANAGER II

All files delivered on the disc/microdrive cartridge enclosed are copyright of Jochen Merz; the files PTR_GEN, WMAN, CONFIG and HOT_REXT are exceptions: they are copyright of QJUMP and have been included under license. If you do not know how to use one of this programs/extensions please have a look at the sections at the end of this manual.

Please make your own backup of the files to work with. These backups are for your own use only!!! Do not modify the master disc/cartridge!

Under no circumstances will Jochen Merz Software be liable for any 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 the software. If you find any bugs in it we will try to fix it as soon as possible. You get free updates if you send the master disc/cartridge together with 2 international reply coupons to

Jochen Merz Software
Im stillen Winkel 12
4100 Duisburg 11
West Germany

In the following manual you can exchange flp to mdv if you own a microdrive version. Parameters shown in square brackets mean options. Some examples use some of the facilities of SuperToolkit II and HOTKEY System II.

The EPROM Manager

The EPROM Manager enables you to put one or more files of the same or different type into one file, so that the whole file can be loaded in one go. The whole file may be put into one or more EPROMs if you wish.

Loading a file containing many files not only saves memory (remember, RESPR takes multiples of 512 Bytes), but it works much faster, especially if you load from microdrive.

First you create a control file. You can use any ASCII-texteditor (for example Metacomcos, The Editor, or better use QD!!!) to do that. This control file should contain any instructions for the EPROM Manager to create the file you wish. After creating the control file execute the EPROM Manager and enter the name of the control file. The EPROM Manager does its work automatically now and either reports an error or gives you the file you asked for. The control file may contain the following commands:

ROM outputfile,size[,copyrightmessage]

tells the EPROM Manager you wish to get a file to put into EPROM. outputfile will be created by the EPROM Manager. size is the maximum work space of the EPROM Manager in kilobytes. Optionally you can give a copyright-message, maximum 35 characters long. EPROM Manager automatically appends a 'NEWLINE'. You know copyright-messages if you own SuperToolkit II or a disc controller, for example. Example:

```
ROM flp1_example,64,Copyright Jochen Merz 1988
```

RAM outputfile,size

is a command to tell EPROM Manager that you wish a file which you want to LRESPR. The parameters are the same as ROM.

Note that the first command of a file must be a ROM or RAM command and that there must be only one of these commands!

There are different kinds of files you may wish to put into a file to load or initialise at a later date. There are many resident extensions, for example RAMPRT (QJUMP's RAM-disc), PTK_BIN (Pointer's Toolkit), HOTKEY System II or PTR_GEN, WMAN, etc. You have to decide whether the resident extension should be initialised immediately when booted or loaded or if it should be invoked with a new SuperBASIC command. It would be very helpful to initialise all the useful things at the beginning of a QL session, but there are some programs which do not like any extension. RAM-discs usually present no problems, but there are some programs which do not work with an initialised Pointer Environment.

So you have to decide now which extension should be initialised immediately. No program worries about RAMPRT, PTK_BIN should be ok, HOT_REXT too. So you can initialise them immediately. You use the command

BIN filename

These files will be initialised immediately when loaded or during boot. Examples:

```
BIN flp1_ramprt
BIN flp1_PTK_BIN
```

Let us now have a look at extensions which will be invoked at a later date. PTR_GEN is one of these such programs, WMAN can be initialised only if PTR_GEN has been initialised. Therefore we add two new SuperBASIC commands which invoke the Pointer Interface and the Window Manager if entered. The syntax is

CMD filename,command

The new command may be up to 14 characters long. You should define it in upper case, otherwise you would get a mixture of upper and lower case commands in a listing. Examples:

```
CMD flp1_ptr_gen,POINTER
CMD flp1_wman,WMAN
```

To install the complete Pointer Environment use, if the created file has been loaded, the new SuperBASIC commands

```
POINTER:WMAN
```

You may create a file with another EPROM Manager control file which puts both files into one file so that you can install the Pointer Environment with one command.

Files like this may be declared to be a Thing but there is no sense at the moment in doing that, as the concept of Things is new and there is no resident extension which supports it.

Most of the other programs you may wish to put into EPROMs are executable programs, jobs. You usually start those kind of programs with EX, EXEC or EXEC W (i.e. programs of filetype 1). Again here you have two options: You can assign a new SuperBASIC command to invoke a job or you can create a Thing to be executed with a Thing command or with a HOTKEY.

JOB filename,command

Is similar to the CMD-command, but this time no extension will be activated, a job will be created. You should use ROM-able jobs only!

THG filename,thingname

declares a given file to be a Thing. The filetype will be the Thing-Type. If the file has a filetype 1, you may combine it with HOT_THING with a HOTKEY. After loading or initialising a file or an EPROM which contains a THG-definition, you get some new things: If before the Thing gets initialised a HOTKEY System II exists, the Thing will be linked into the Thing-list automatically.

In any case you get a new SuperBASIC command, TH_ plus the Thingname, to link in the Thing at a later date or if it has been removed with TH_REMV or TH_FRMV. Notice that you can use all these facilities only if you have a HOTKEY System II V2.03 or higher version, otherwise it is impossible to link in Things.

Thing names must be at least three characters long, but a maximum of 12 characters is allowed.

You can put a thing directly on a HOTKEY, even in ROM! First you have to declare the Thing, then you declare the HOTKEY. The command is

HOT_EXEC key,thingname

To put QRAM on a HOTKEY '/' (as always), you could put the following two lines into you EPROM controlfile:

```
THG flp1_QRAM,QRAM
HOT_EXEC /,QRAM
```

After you burned the ROM and started the computer with the ROM you will find that a HOTKEY '/' exists. Have a look with HOT_REXT's HOT_LIST command.

Finally you can declare files to be a device. After RESET the QL looks for a device 'BOOT' and, if not found, for a 'MDV1_BOOT'. So you can make your own BOOT if you wish. You just write your own BOOT-program which does all the things you want after RESET, remove the line numbers (very important!!! QD helps you do this!), and declare it to be new device BOOT.

DEV filename,devicename

You can use any other devicename, of course, up to a length of 4 characters. Beware of over-defining standard devices like con, scr, ser, pipe or par, for example!

If you declare a procedure to be a device, e.g.

```
DEV mdv1_someprocedure,PROZ
```

you can have a look at it at any time by typing in

```
COPY PROZ TO scr
```

or you can load or merge it in the usual way to execute it immediately.

An example: You wish to put the major things of your BOOT-file into EPROM. This boots much faster and leaves you more RAM space to work. A control file for doing this would be

```
ROM flp1_BOOT_ROM,64,My BOOT-EPROM
BIN flp1_RAMPRT
BIN flp1_THING_rext
BIN flp1_HOT_rext
CMD flp1_PTR_GEN,POINTER
CMD flp1_WMAN,WMAN
BIN flp1_QLIB_run
THG flp1_QRAM,QRAM
THG flp1_SYSMON,SYSMON
```

You save this control data to a file, say

```
flp1_BOOT_CTL
```

Now you can start the EPROM Manager:

```
EX EPROM_MANAGER
```

and enter the control-filename:

```
flp1_BOOT_CTL
```

All further action is automatic. You will get a ready file presented or you will read an error message. Be sure, all the files listed above do not fit into one 64k EPROMs, so you have to give at least 128k and put it into two or more EPROMs.

If you burned the file into one or two EPROMs, you could make a BOOT file:

```
10 ERT HOT_THING('/',QRAM):REMark QRAM
20 POINTER:WMAN:REMark gives us Pointer
30 TH_EX SYSMON:REMark executes SYSMON
40 HOT_GO
```

This file would replace a BOOT file of the following form:

```
10 a=RESPR(6448):LBYTES flp1_RAMPRT,a:CALL a
20  a=RESPR(2998):LBYTES  flp1_THING_rext,a:CALL
    a
30 a=RESPR(10946):LBYTES flp1_HOT_rext,a:CALL a
40 a=RESPR(13448):LBYTES flp1_PTR_GEN,a:CALL a
50 a=RESPR(7882):LBYTES flp1_WMAN,a:CALL a
60 a=RESPR(10064):LBYTES flp1_QLIB_run,a:CALL a
70 ERT HOT_CHP('/',flp1_QRAM)
80 EX flp1_SYSMON
90 HOT_GO
```

General Rules

You should never put impure or any code which is not ROM-able into ROM.

Some resident extensions need a special sequence. Some extensions do not like to be loaded before others, some do not worry about their position in the initialisation sequence.

Never load PTR_GEN before SPEEDSCREEN or LIGHTNING!

Never initialise Toolkit II after HOT_REXT! If you have to invoke Toolkit II with the command TK2_EXT, you should define a command for HOT_rext, say HOT_EXT. After the QL started up, you enter TK2_EXT:HOT_EXT.

ROM-ABLE OR NOT?

You must not put programs or extensions into ROM which are not ROM-able. You can check if a program or extension is ROM-able if you use the new commands of the extension EPROMTEST_rext. Load the extension the normal way, i.e. with LRESPR or the boring RESPR LBYTES-CALL sequence. You get two new commands:

ROM_LOAD filename

loads an EPROM-file and initialises it in the same way as if really is in EPROM. Firstly the ROM-name will be printed out, then any new commands (if there are any) will be initialised, after that the EPROM initialisation (if there is any) will be executed. After doing that a checksum will be generated and the contents of the 'EPROM' should now work. You can test everything. Try to do some things again. If everything works without a crash, your chances are quite good. You can enter now

ROM_TEST

and you will immediately know if the EPROM code has been modified by itself (not very good!). There is no warranty that the EPROM file is 100% rom-able, but you can assume that it is.

So, here you see how the file would behave in ROM, so it's not necessary to burn it to see if there is anything wrong.

After doing a ROM_LOAD and a ROM_TEST you should RESET your QL.

SPLIT FILE

This is a useful utility for large EPROM files. If you have files longer than 64k and you wish to burn them into EPROM you will surely run into some trouble. Most memory expansions do not work with the QEP III, so you have only 64k workspace for the QEP III. You are lucky if you own an internal memory expansion, this will work with QEP III. Another case is, if you have a simple EPROMmer and you would like to put a 64k file into 2 27256 EPROMs. You have to split the file into two files of the required length. SPLIT FILE helps you do this. It splits any file into two or more parts to fit into any available EPROMs. First you have to start the program:

```
EXEC flp1_SPLIT_FILE
```

Enter the filename of the EPROM file, e.g.

```
flp2_TEST_ROM
```

and select the size of the EPROMs, in the example 4.

You will get two files now,

```
flp2_TEST_ROM_0
flp2_TEST_ROM_1
```

(if the file is longer, the next file would be flp2_TEST_ROM_2 and so on)

THING EXTENSION II

You will find the following procedures and functions in this manual:

Things

First of all you have to know what things really are, as the name tells you nearly nothing.

Things are general-purpose resources which may be used by any code in the system, either from device drivers or directly from programs. Things may be shareable by a finite or infinite number of 'users'.

A Thing normally has a version number. It is declared either directly (if the Thing is declared to be a Thing, e.g. HOTKEY System II) or, if it is loaded at a later date and declared to be a Thing (e.g. TH_LOAD) it will get the version number found in the header of a configuration block (if one exists). This means: every program which has a standard configuration header will have the same version number to be the Thing version.

Things are identified by name (like files). Executable Things (e.g. files, whose filetype is 1, which may be EXECuted) may be started from RAM or ROM. You do not have any loading time and need less memory. Each execution of an executable Thing will not copy (or load) the whole program into RAM, it just creates a new job header. Every running copy of that thing shares the same code, but has a different data-area.

The above applies to all other kinds of files. Once declared to be a thing it is possible to find it in RAM or ROM. Any piece of code can find it. To give an example think of user-defined-character sets. If you start three or four compiled SuperBASIC-Jobs which use that character set, every program will load it and use it itself. There was no easy way to let other jobs know where the character set has been loaded.

Things give you an easy way to do it. Everything, which has been declared to be a Thing may be used from any code or job running in the machine.

A piece of code that wants to use a Thing supplies the System with the name of the Thing. The result returned is the address of the Thing. If the call to use a Thing is successful, the job is said to be a user of that thing. This job may free a Thing at a later date. A Thing will also be freed if a 'user'-job disappears.

If a Job, which installed a Thing, disappears, and the Job is the owner of the memory occupied by the Thing, the Thing will also disappear. This can lead to further action: All jobs which use the Thing at the time it is removed have to be removed also, as their existence is based on the existence of that Thing. A special case is SuperBASIC: SuperBASIC itself must not be removed, so it is more difficult to remove Things created by SuperBASIC. The result is: any Job should use a Thing only for the time it really needs it and then free it.

Removing a Thing which is not in use gives no problems (TH_REMV). The memory occupied by that Thing will be released. In this case no job will be killed as there is no job using the Thing. You can force remove Things in SuperBASIC (TH_FRMV). All jobs which currently use the Thing will be removed except the Thing which is doing the force remove. A very complicated example would be compiled SuperBASIC, which has been started by a job which is removed when the Thing is removed. This means all jobs which are owned by jobs which disappear would also have to disappear. In this case the compiled SuperBASIC would also be removed.

This may sound very complicated, but it is not. All this ensures that QL's memory will be cleared if there is something removed to the state before.

The Thing-Extension

The Thing-Extension gives you some SuperBASIC commands and functions to control and use Things. You can load the Thing-Extension by typing

```
LRESPR flp1_THING_rext
```

or for people without SuperToolkit II

```
a=RESPR(size):LBYTES flp1_THING_rext,a:CALL a
```

THINGS [#channel]

gives a list of all currently existing Things. Default is channel #1, but you can give any other channel of course.

You will see a table of three columns: the version-number (if there is any) of the Thing, the type and the name of the Thing.

The version-number is either the real version-number set by the Thing itself or a version-number found in the Configuration-Block (if there is any) if the Thing has been loaded with TH_LOAD. Things installed with the EPROM Manager get a version 'ROM' instead of the real number (this applies for RAM-files made with EPROM Manager too).

The Thing-type is (nearly) identical with the usual file-type. TH_LOAD takes the file-type and declares it to be the Thing-type. The following types exist at the moment:

-1	VECT	special VECTOR THING.
0	UTIL	file-type 0 (if loaded) or UTILITY
1	EXEC	file-type 1 (if loaded), executable
2	DATA	thing containing any type of data
3	EXTN	thing contains or is an extension

If the type is 3, you will find a list of the extension IDs defined in that thing. An example of extension thing is the ser_par_prt Thing of the QL Emulator for the ST or the Menu Extension.

The special THING is the only Thing which has upper-case letters. Thus you cannot use or remove it. It contains vectors to access Things from machine-code.

The Thing-name is very analogue in its form to file-names. It does not matter whether you specify the Thing-name in upper or lower case or a mixture of it. You can see it in the list in lower case anyway, in order to speed up comparison. There is one restriction: Thing-names have to be at least three characters long. This also gives faster comparison. If you TH_LOAD a file name shorter than three characters, one or two '_' will be appended.

TH_LOAD filename [,thingname]

loads in the given file and declares it to be a Thing. The name of the Thing is the name of the file without the drive. You may give a drive specifier. If you do not, the data default directory will be used. If the name is shorter than three characters, '_' will be appended to give a name of three characters length. The Thing-type will be the same type as the file. If the file contains a standard-configuration-block the version number will be used to give the Thing-number. There may be an optional name if the Thing name has to be different to the file name.

Examples:

TH_LOAD flp1_QUILL
defines a Thing type 1 (executable) named quill.

TH_LOAD flp2_qd
defines a Thing type 1 named qd_.

TH_LOAD mdv1_qd,Editor
defines a Thing type 1 which is QD, but the name should be Editor.

All usual file errors may be returned. If a Thing with the same name already exists, an error 'already exists' will be returned.

TH_REMV thingname

removes a Thing. The whole memory occupied by the Thing is released.

Possible error returns: The Thing does not exist or it may be used by a job.

TH_FRMV thingname

removes a Thing, even if it is used by one or more jobs. All jobs using it will be removed.

TH_USE (thingname [,timeout])

is a function. It return the address of the Thing or a negative number which is an error code. You can convert it into a message with REPORT. If an address is returned, the current job will be a user of the thing, until the job or the Thing is removed or the job frees it with TH_FREE.

An optional timeout may be specified. Timeouts are useful if the Thing which should be used is not shareable and currently in use. The default timeout is 'forever', so you have to BREAK. If you specify a timeout, TH_USE will return after the given time if the Thing is still in use.

The address of the Thing is the start including header. You will find here always the standard identification 'THG%'. To find the start of the code loaded in with TH_LOAD, you have to add the value TH_OFFSET.

To use a TH_LOADed font, use

```
address=TH_USE(fontname)+TH_OFFSET
```

Another example:

```
10 adr=TH_USE(tra_table)+TH_OFFSET
20 IF adr<0
30  REPORT adr
40 ELSE
50  PRINT 'TRA TABLE at address '!adr
60 ENDIF
```

TH_FREE thingname

frees a Thing. The job will be taken out of the list of users of that Thing. If you are not using the Thing you will get 'not found'.

TH_USER [#channel,] thingname

gives you a list of all jobs which currently use the given Thing. You may specify an output-channel, default is #1. You get a list of jobnumber, tag and job-name for every job using it.

TH_EX thingname [;parameter$] [,priority]

executes an executable Thing (type 1, EXEC). An optional parameter string may be passed to the job. This parameter string has the same function as defined in the EX command of SuperToolkit II. You can give an optional priority, otherwise 8 will be used. Note that many programs change their priority directly after the start.

Note also that only a new data area will be used by that job; there may be many jobs running which share the same program. This can save an enormous amount of memory. Important: executable Things must not modify the program code, otherwise only one copy of that must can be executed!

Examples:

```
TH_EX Editor
```
executes a Thing 'Editor'

```
TH_EX Grabbed_Quill;"100"
```
executes a grabbed Quill and allows a maximum workspace of 100k Bytes.

```
TH_EX asm;'flp2_test -errors',100
```
starts an assembler, sets its priority to 100 and passes a parameter string.

```
TH_EX 'clock',1
```
starts a clock and sets the priority to 1.

TH_TYPE (thingname)

is a function which returns the type of a Thing (or an error message, as always). TH_TYPE has to use a Thing to find out its version number. The timeout is 2 seconds.

TH_VERS$ (thingname)

is a function which returns the type of a Thing (or an error message). If the Thing does not have a Version number, an empty string is returned. TH_VERS$ has to use a Thing to find out its version number. The timeout is 2 seconds.

Using Things

A Thing can be thought of as a kind of file in RAM and/or ROM. One or more Jobs may access Things, similar to shared files. By way of example, let us use the SuperToolkit II-command

```
CHAR_USE #channel,font1,font2
```

to assign new fonts to a SuperBASIC channel. But, first you have to get the fonts into memory somewhere. Let us assume that the length of the font is 850 bytes. You have to get 850 bytes memory space. Normally you get memory space by RESPRing it. This works if there are no jobs running (a rare occasion in my machine) and this has the disadvantage that you cannot release the memory to use it for other things (not Things). ALCHP also has some disadvantages: when doing LOAD, NEW etc. the memory will be released to the system automatically, after CLEAR the address of the font gets lost ... In either case you have to re-load the font.

A further disadvantage is: if there are some compiled jobs all needing the same font, all have to load them just for their own use. This does not happen with Things! First you load a font (TH_LOAD) and declare it to be a Thing. After doing that any job can access this font with TH_USE. If you do NEW, CLEAR or LOAD, you can get the address at any time just by using it again. You do not have to re-load the font.

A further advantage in using Things is the ability to held programs constantly in memory and to access them with HOTKEYs. That's what HOT_CHP and HOT_RES (of HOTKEY SYSTEM II) do: they load the file and declare it to be a Thing. You do not believe it? HOT_CHP something and have a look at the Thing-list.

You will find that HOTKEY II itself is a Thing, and, if it has a version number V2.03 or higher, you will find a Thing called 'THING' too. This THING is an exception, because it is defined upper case and cannot be used or removed.

If you have a Thing in ROM or RAM, you can combine it with a HOTKEY. This may be done with the HOT_THING function. Let us assume you made yourself some EPROMs with the EPROM Manager or you loaded a Thing with TH_LOAD, you can combine it with a hotkey, e.g.

```
ERT HOT_THING(key,thingname)
```

After that you can press the given key to invoke the program.

There is a problem when you remove the HOTKEY: The Thing combined with that key will also disappear, but if you remove the Thing, the HOTKEY will not be removed. If you remove the HOTKEY after you removed the Thing you will get a message 'not found'. This does not mean that the HOTKEY does not exist (perhaps it really does not exist), it can mean that the Thing combined with the HOTKEY does not exist.

The extension Things (type EXTN) also have many advantages: the interface to the command language (e.g. SuperBASIC) is much easier than the existing one. If you remove the extension, the interface makes sure that it is really 'clean' removed, e.g. no crash if you remove a command which does no longer exist. And, if there are a lot of BASICs (future ...) running around and using this extension, the removal of the Thing makes sure that the BASICs which use them are also removed.

POINTER ENVIRONMENT

It is very easy to explain, but adds great power to the QL. The Pointer Environment consists of two resident extensions, called PTR_GEN (an extended version of the Sinclair QL Pointer Interface) and WMAN (the Window Manager). The window manager will be used only from programs which are programmed to do so. The Pointer Interface will add facilities to the QL which can be used by many programs, even programs which were written before the Pointer Interface existed.

You load in the resident extensions in the same way as other resident extensions:

```
LRESPR flp1_PTR_GEN
LRESPR flp1_WMAN
```

If you do not have a SuperToolkit II, you have to use

```
a=RESPR(15000):LBYTES flp1_PTR_GEN,a:CALL a
a=RESPR(9000):LBYTES flp1_WMAN,a:CALL a
```

The sizes used in this example are also examples. You have to find out the real sizes and use them instead. You should add these lines to your BOOT program!

The Pointer Interface adds 'real' undestructive windows to the QL. CTRL C does not only move the active Cursor, it moves all windows of a job. So, do you have some jobs handy? Start a job and press CTRL C. Press CTRL C again. Do you see the effect?

In menu-driven programs you can select items by pointing to them and pressing SPACE or ENTER, or by pressing the first letter of the text within that item. Try loading a program which uses the Window Manager.

Now move the pointer. Move it over a 'Menu item'. You will see, that it gets a border as soon as you touch it. This is the 'current' item. Now press SPACE. The menu item will change its colour, maybe something happens (depending on the item).

SPACE always selects or deselects, depending on its state. You may have pressed ENTER, this selects and executes. SPACE and ENTER may have the same effects, but not always. You may, if you own a mouse (connected to an ST with QL Emulator, QIMI or SuperQBoard) press the left mouse button instead of SPACE or the right mouse button instead of ENTER. The third way of selecting an item is pressing the first letter of its name (sometimes a function key etc). The rule is: point to it and select or press the first letter.

There are special items which have special keystrokes.

ESC		Escape, Quit, Abort
F1	is	Help
CTRL F1	is	Sleep
CTRL F2	is	Wake up button or update window
CTRL F3	is	Window resize
CTRL F4	is	Window move

Not all programs offer all the facilities!

Move the pointer out of the window. You will see, it appears to be lock. This means: there is a window underneath which is locked. Now press SPACE or ENTER. You will 'pick' this window. It is easier then doing lots of CTRL C's.

The Pointer Environment offers many facilities, the Window Manager even more. To mention them all would fill many pages. You will find special facilities used by a program explained in the manual of that program.

CONFIGURATION

To start CONFIG use

EXEC flp1_CONFIG or
EX CONFIG

Using CONFIG is very easy. CONFIG asks for the file to configure. These files may contain one or more configurable parts.

Each part has its own name and version number. You will be asked in turn if you wish to configure each of the parts. Press ESC when you have configured everything that you wanted.

Once you have said YES to configure a program, there will always be some text to which you have to reply. You can always press ENTER to go to the next question, press ESC to quit or answer the question.

You will be prompted for different types of parameters, e.g. string, number, a single character or a number of pre-defined options.

You can go through many options by pressing SPACE. When you found the right option, press ENTER.

You may press ESC at any stage to save the setups you have done so far. Press ESC again to configure another program or a press it a third time to leave CONFIG.