THING Extension V3

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

The Thing-Extension gives you some SuperBASIC commands and functions to control and use Things. The HOTKEY System II has to be present, otherwise the Thing System does not exist at all (it is part of the HOTKEY System II). You can load the Thing-Extension by typing

```
LRESPR flp1_THING_rext
```

or for people without SuperToolkit II (size is the size of the file)

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

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

THINGS [#channel] or
THINGS \device

gives a list of all currently existing Things. Default is channel #1, but you can give any other channel of course. If #1 is not open, then #0 is used instead.

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 contains 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.

If the channel where the listing is sent to is a SCR or CON channel, then output freezes after a screenful of things. Press any key to continue.

This version of the THING-Extension allows implicit channel handling. Maybe you already know it from SuperToolkit II. Examples:

```
THINGS                          diverts output to screen channel #1.
THINGS #2                       outputs to channel #2 (usually screen).
THINGS \flp1_allthings          lists to file FLP1_ALLTHINGS.
```

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. If it is not open, channel #0 is used instead. You get a list of jobnumber, tag and job-name for every job using it.

TH_EX thingname [;parameter$] [,priority] or

TH_EW 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!

TH_EW is the thing-equivalent of EW or EXEC_W, which means, it has the same function as TH_EX, but waits until the new job has terminated.

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_EW asm;'flp2_test -errors',100
```
starts an assembler, sets its priority to 100 and passes a parameter string. It waits until the assembler has done its job.

```
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 code). 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.

TH_CREATE thingname,size

creates a Thing for your own purpose. You can use it to pass data between jobs or for storing data for a number of jobs. The usable size is given in bytes, and you can find out the base address of this area by using

```
PRINT TH_USE(thingname)+8
```

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.