QJUMP HOTKEY SYSTEM II

HOTKEY	Extension
ERT	command
HOT_RES	function
HOT_CHP	function
HOT_LOAD	function
HOT_THING	function
HOT_PICK	function
HOT_KEY	function
HOT_CMD	function
HOT_STUFF	function
HOT_GO	command
HOT_STOP	command
HOT_LIST	command
HOT_NAME$	function
HOT_TYPE	function
HOT_OFF	function
HOT_SET	function
HOT_REMV	function
HOT_DO	command
HOT_WAKE	command
HOT_RES1	command
HOT_CHP1	command
HOT_LOAD1	command
EXEP	command

The HOTKEY system has been extended beyond the original concept of a system for activating copies of resident programs to include the SuperToolkit II ALTKEY and Last Line Recall (ALT ENTER) facilities, "picking" jobs to work with them, executing programs from disk or microdrive and issuing commands to the SuperBASIC interpreter.

HOTKEY SuperBASIC Extensions

The HOTKEY system includes a number of SuperBASIC extensions to enable the HOTKEY system to be manipulated from SuperBASIC. Most of these extensions are in the form of functions: this enables error checking to be carried out simply, and any corrective action taken. All of the

HOTKEY extensions start with "HOT_" so you should have no problem identifying them.

Using the HOTKEY system involves three stages: first the HOT REXT resident extensions file should be loaded and called (either using the Toolkit II command LRESPR or using the time honoured RESPR, LBYTES and CALL statements). Next any programs required are added to the HOTKEY system using these resident extensions. Then the HOTKEY system is activated. This starts the HOTKEY job which will do very little until you press an ALT key combination. When you press an ALT key combination which has been set up as a HOTKEY, the HOTKEY job will leap into action, and do whatever has been specified. If the attempt fails (possib1y because there is not enough memory) HOTKEY will burp and retire into the background again. Most HOTKEYs will be set up in a BOOT file, but you can add, remove or change any HOTKEYs at any time.

Errors and Defaults

The functions used to set up, change and remove HOTKEYs have two distinct error handling methods. If the function is used incorrectly, (e.g. missing parameters), then execution of the program will stop in the usual way. If, however, the parameters are correct, but you are trying to do an invalid operation (e.g. redefining a HOTKEY without removing it, or trying to load a file which does not exist), then the function will return an error code for further processing. One such code which can be returned from any of the functions is ERR.IU (-9, in use) which can occur if a program has tied up the HOTKEY system for more than 2 seconds. If there is a long pause before an "in use" error return, this is the reason.

Many of the functions can be supplied with filenames. It is not necessary to specify a drive name as the HOTKEY system has its own default built in. This default can be changed by the CONFIG program supplied. If you have a Toolkit with an executable program default (e.g. SuperToolkit II) then this default will be used instead.

In general all the parameters of a HOTKEY function can be given as either "strings" or "names". A name must start with a letter, and contain only letters, digits and underscores. A string can have any characters between apostrophes or quotes. If in doubt put the parameter between quotes or apostrophes: particularly if you will be compiling your program.

Furthermore, when defining the HOTKEY itself. the key is best placed between apostrophes or quotes to avoid problems with the SuperBASIC name handling which does not distinguish between upper and lower case.

Case Dependent HOTKEYs

You can define HOTKEYS in two ways. If you define a lower case HOTKEY, then the HOTKEY action can usually be invoked by pressing ALT and the appropriate letter, regardless of whether the SHIFT key is pressed or CAPSLOCK is set.

Alternatively, if you define an upper case HOTKEY, then this action will only be invoked by ALT and the upper case character.

For example, if these HOTKEYs are set:

HOTKEY	ACTION
A	EXEC Alarm
Q	EXEC Quill
Q	EXEC QRAM

"ALT Q" (ALT SHIFT Q) will execute Quill, while "ALT q" will execute QRAM. Both "ALT A" and "ALT a" will execute the alarm clock.

Error Reporting

Because so many of the extensions are defined as functions, it would be useful to be able to use them as procedures as well. A boot file (or other program) would then stop automatically with the usual cryptic messages. Unfortunately this cannot be done directly with the standard SuperBASIC interpreter, but the HOTKEY system includes a simple procedure which will report the error and stop if its parameter value is negative. This procedure, ERT, can be used with any function which returns an error code (e.g. many of the Qtyp spelling extensions) as well as with the HOTKEY extensions.

```
herr=HOT_RES('t',flp1_qtyp)    herr is error from HOT_RES
PRINT HOT_RES('t',flp1_qtyp)   print error from HOT_RES
ERT HOT_RES('t',flp1_qtyp)     stop if error from HOT_RES
```

Adding a Program to the HOTKEY System

A program may be added to the HOTKEY system using one of the functions

`HOT_RES(`*`key,file name`*`)` *load into resident procedure area*
`HOT_CHP(`*`key,file name`*`)` *load into common heap*

HOT_RES should normally be used, but if there are any jobs executing in the QL, it will fail. If this happens, it is automatically converted to HOT_CHP. If you wish to add a program temporarily, then you should use HOT_CHP. You may then remove it at any time (see HOT_REMV).

HOT_RES and HOT_CHP can only add executable (compiled) programs to the HOTKEY system. Interpreted SuperBASIC programs can be loaded and run using commands set up by the HOT_CMD function (see below).

Programs added using HOT_RES and HOT_CHP are resident in the QL, and copies of these programs are instantly available. For less used programs, it would be better to use the HOT_LOAD function (see below) which loads the program from disk or Microdrive as required.

The key is a single character or single character string defining the HOTKEY which will invoke the program. The file name can be a name or a string. If you are using SuperToolkit 11, then the program default directory will be used. Otherwise, the HOTKEY system will use its own default.

HOT_RES and HOT CHP return the value zero, or a (negative) error code.

The error returns that can be expected are

ERR.NJ	-2	file is not executable
ERR.OM	-3	out of memory
ERR.NF	-7	file not found
ERR.IU	-9	key already defined or file is in use
ERR.BN	-12	bad file name

Here are some examples of adding QTYP to the HOTKEY system. Any other well behaved software may be added in the same way. The fifth example shows how it is possible to detect that the attempt to load a resident program has failed and to recover from this.

```
ERT HOT_RES('t',qtyp)           with default drive

ERT HOT_RES('t',flp1_qtyp)      or specified drive

ERT HOT_RES('t','flp1_qtyp')    or all between apostrophes

ERT HOT_CHP('t',qtyp)           so we can HOT_REMV it
```

```
REPeat lqtyp
 herr = HOT_RES ('t' , 'qtyp')              try loading Qtyp
 IF NOT herr: EXIT lqtyp                    ...OK
 IF herr = -7                               not found?
  INPUT #0,'Put Qtyp disk in drive 1 and press ENTER'
  NEXT lqtyp                                try again
 END IF
 PRINT #0, 'Loading Qtyp '; : ERT herr      give up
END REPeat lqtyp
```

Each program added to the HOTKEY system is identified by a name. Normally this will be the program name taken from the base area of a standard program. It is possible, however, to give the HOTKEY program a name which is different. For reasons which may become apparent later, this name should be longer than 3 characters.

```
HOT RES (key, file name, program name)
HOT-CHP (key, file name, program name)
```

For example, a specially configured version of QTYP could be added to a different keystroke from the normal "T".

```
ERT HOT_RES ('=', qtyp_e, 'Editor Qtyp')
```

Badly Behaved Programs

Some programs are badly behaved in some ways. There are variations to cater for two of the most common misdemeanours: impure code and

grabbing most of the memory. Impure programs can be added to the HOTKEY system by adding the single parameter "I" (upper or lower case) to the function parameter list. Before the program is started, a copy is made of the code. This ensures that the original code remains unmodified. Note that this means that whereas pure HOTKEYed programs will have only one copy of the code in memory, however many copies of the program are executing, impure HOTKEYed programs will have one more copy of the code than there are copies executing. It might be better to HOT_LOAD programs of this type unless you have an excess of memory in your QL. Using this variation, even BCPL and Turbo compiled programs can be added to the HOTKEY system. You should not specify a program name for impure programs: this could cause problems.

```
HOT_RES (key, file name, I)
HOT_CHP (key, file name, I)

ERT HOT_RES ('e',flp1_edt_bin',i)  adds The Editor (ALT E)
```

The Psion programs have the nasty habit of grabbing most of the QL's spare memory to prevent other jobs from running. A special variation is used to reduce this unpleasant effect, by being even more unpleasant. The HOTKEY program will grab most of the memory itself, just leaving enough for the Psion program, and when the Psion program has started, gives it back again, which is ever so nice of it. It is possible to specify the memory you require to be left for the program (about 32k is usually adequate). If you do not, then every time the program is started, the HOTKEY program will ask the user the amount of memory to be left. The amount of memory that the Psion program will actually take depends on circumstances, but will always be slightly less than that allowed.

You do not need to use this variation if the Psion program has already been processed by Grabber. Indeed, the thought of having three levels of program (HOTKEY, Grabber's DAEMON and the Psion program) all fighting each other for the privilege of grabbing all the memory, is enough to give QDOS a headache.

This variation adds the letter "P" to the parameters of the various functions.

```
HOT_RES (key, file name, P)      ask amount of memory
HOT_CHP (key, file name, P)

HOT_RES (key, file name, P, memory in kilo-bytes)
```

```
HOT-CHP (key, file name, P, memory in kilo bytes)
```

For example, you can add Quill to the HOTKEY system, loading it into the common heap and allowing it 32 kilobytes of working memory.

```
ERT HOT_CHP (q, Quill, p, 32)
```

Other Variations

There are two other variations on the functions to add programs to the HOTKEY system. These are to allow for the differences between the Pointer Environment and the ordinary QL environment. The first ia that there are some programs which utilise the rather untidy, destructive windows of the standard CONsole device driver. Windows in the Pointer Environment can be made destructive by "unlocking" them. The second is to allow for jobs which do not have a window which covers the whole area used by the program. If you need to use one of these variations, this does not imply that there is anything wrong with the software.

To unlock the windows of a job in the HOTKEY system, you need to add the single parameter "U" to the function parameter list. To provide a "guardian" window to preserve the whole area used by the job, you need to add the single parameter "G" to the function parameter list. Optionally, you may follow this by the window area (size, position) of the guardian window as four numbers. Any attempt by a program to open or redefine a window outside its guardian will fail. Note that either "U" or "G" can be used after the "I" option for impure programs.

```
ERT HOT_RES (c, capsc1ock, u)
```
add unlocked "capsclock"

```
ERT HOT_RES (x, text87, g)
```
add TEXT87 with guardian window covering the whole screen

```
ERT HOT RES (q, text87, Quill, g)
```
add TEXT87 with guardian and call it Quill!!

```
ERT HOT_RES (r, rubbish, i, g, 124, 22, 388, 0)
```
add "rubbish", an impure program which requires a guardian 124x22 pixels with its origin at 388x0

Loading and Executing Files from a HOTKEY.

If a program is not required frequently enough to justify making it resident. it is possible to define a HOTKEY to load and execute the program from disk or Microdrive. This is similar to the HOT_RES and HOT_CHP, but the program is not loaded until required. It follows, of course, that the disk or Microdrive with the program file must be available at the time you press the HOTKEY. The Impure variation should be specified for impure code such as BCPL or Turbo compiled programs, although this is ignored in this version. The "Psion" variation is available to execute Quill, Archive, Abacus and Easel, as are the "Unlock" and "Guardian" variations.

The function HOT LOAD returns the value 0 (ok) or -9 (ERR.IU) if the HOTKEY is already defined or the HOTKEY table is full.

```
HOT_LOAD (key, file name)
HOT_LOAD (key, file name, P)
HOT-LOAD (key, file name, P, memory in kilo bytes)
HOT_LOAD (key, file name, U)
HOT_LOAD (key, file name, G)
HOT-LOAD (key, file name, G, window definition)
```

For example, you can set up to load and execute Qtyp_file every time you press ALT F and Abacus (with 60k memory allowed) every time you press ALT A:

```
ERT HOT_LOAD (f, qtyp_file)
ERT HOT_LOAD (a, abacus, p, 60)
```

Adding a THING to the HOTKEY System

You can add an executable program which is defined as a THING to the HOTKEY system. The Thing need not be defined at the time it is added. Do not worry too much about Things, they seldom go bump in the night, and few QL users have ever met one.

```
HOT_THING (key, thing name)
```

QRAM II is implemented as a collection of (mostly) executable Things. The HOTKEY system itself creates an executable Thing for each HOT_RES or HOT_CHP call. The HOTKEY system is a non-executable Thing.

Picking a Program

HOT_RES, HOT_CHP, HOT_LOAD and HOT_THING all set up HOTKEYs to execute a new copy of a program. Each time you press one of these HOTKEYs you create a new job. On the other hand, HOT PICK sets up a HOTKEY to pick an existing job, so that you may work with it. The job is identified by the program name. The HOT_PICK function can return either 0 (ok) or -9 (ERR.IU)

```
HOT_PICK (key, program name)
```

The program name is usually embedded at the start of the program file. For pure programs set up by HOT_RES and HOT_CHP, this name is overwritten if you specify a program name. For Psion programs, which do not have a name at the start, the HOTKEY system uses the name of the file.

When the HOTKEY system tries to find a job to pick, it will accept an abbreviation of a program name, provided that the first character after the abbreviation is not a letter. (e.g. PICKing "QTYP" will pick "QTYP", "QTYP2", "QTYP_E" but not "QTYPER"). If there is more than one job executing with the same program name, then each job will be picked in turn.

```
ERT HOT_PICK ('1',Quill)        pick Quill on ALT 1
ERT HOT_PICK ('2',Abacus)       ... Abacus on ALT 2
```

Stuffing the Keyboard Queue

SuperToolkit 11 users will probably be familiar with the ALTKEY and Last Line Recall facilities. These facilities are reproduced in the HOTKEY system. One difference is that the stuffing is done by a Job rather than the Polling task used by SuperToolkit 11. This is not so versatile, but it does mean that the hiccups that occasionally occur in the Last Line Recall of SuperToolkit 1I should not happen.

The HOTKEY system also has the stuffer buffer of the original HOTKEY system, but in an extended form. As well as stuffing the last stuffer string set by QRAM, the Calculator and other utility software (using the HOT_STUFF command or similar), you can stuff the previous string (and the one before that ....).

To recall the last line typed in any window, press ALT and ENTER. To stuff the current keyboard queue with the current stuffer buffer string, press ALT and SPACE. To go back to the previous string, press ALT, SHIFT and SPACE. The keystrokes for the stuffer buffer, as well as the stuffer buffer size, can be set by the CONFIG program supplied.

Pre-defined strings can be set up on specified keys using the HOT_KEY function

```
HOT_KEY (key, list of strings)
```

When the appropriate HOTKEY is pressed, each of the strings is sent to the keyboard queue, separated by a newline (ENTER) character. You can specify as many lines as you like. If you want a newline after the last HOT_KEY string, you should put a null string at the end of the list.

```
ERT HOT_KEY(s,'Dear Sir,',"","")     2 newlines at end
ERT HOT_KEY(e,' Yours sincerely',"","","",'Joe Bloggs')
```

SuperBASIC Commands

It is possible to set up one or more commands to be sent directly to the command console of SuperBASIC with the HOT_CMD function

```
HOT_CMD (key, list of commands)
```

When the HOTKEY is pressed, SuperBASIC is picked to the top, and each command is sent to the command console, followed by a newline (ENTER).

This can be used to load and run SuperBASIC programs, or to execute simple command sequences. If you wish to have several SuperBASIC programs resident at once, each should be defined as a procedure, and a HOTKEY set up to invoke each procedure.

```
ERT HOT_CMD (m, 'LRUN flp1_mandel')   LRUN a BASIC program
ERT HOT_CMD (d,wdir)                  SuperToolkit direct listing
ERT HOT_CMD(r,'INPUT"Run> ";prg$', 'LRUN "mdv1_"&prg$')
                      prompt for name of, and LRUN a program, note
                      the use of both quotes and apostrophes.
```

Setting the Stuffer Buffer

The command HOT_STUFF adds a single string to the Stuffer Buffer

```
HOT_STUFF string
```

You should use a string rather than a name with this command to avoid problems if you have a QPTR Pointer Toolkit which uses this s1ightly restricted definition.

Starting and Stopping the HOTKEY System

A number of software suppliers provide resident extensions to the QL that include an executable job (like HOTKEY). Some of these start the job as soon as the resident extension is loaded and called. Although this seems simpler for users, it is very very naughty. Once there is an executable job in the QL, you cannot load any more extensions. The so called fixed versions of the RESPR function supplied with these can actually cause more problems than they solve. The HOTKEY system is designed to remain dormant until all resident extensions have been loaded. It is then activated by the HOT_GO command.

If, at any time, you wish to add more resident extensions to your QL, you can remove the HOTKEY job using QRAM, the Toolkit II RJOB command or the HOT_STOP command.

Neither HOT_GO nor HOT_STOP have any parameters.

`HOT_GO`	*start HOTKEY job*
`HOT_STOP`	*stop HOTKEY job*

Viewing the HOTKEYs

The current list of HOTKEYs can be sent to any channel using the HOT_LIST command, or the program name associsted with any HOTKEY can be
found using the HOT_NAME$ function. The type of action can be found using the HOT_TYPE function

	`HOT_LIST`	*list HOTKEYs to channel#1*
or	`HOT_LIST #channel`	*list HOTKEYs to given channel*
or	`HOT_LIST \file name`	*list HOTKEYs to file*

```
HOT_NAME$ (key)          return program name for HOTKEY
HOT_TYPE (key)           return HOTKEY type
```

The HOT_NAME$ function returns a null string if the name is not defined. This can be used to provide more control over the HOTKEY system from SuperBASIC programs. For example, you can find out whether a particular key is in use, or a version of HOT_LIST may be written in BASIC:

```
FOR chr=32 TO 191
 hkey$ = CHR$(chr)
 hname$ = HOT_NAME$ (hkey$)
 IF hname$<>'': PRINT hkey$, HOT_TYPE (hkey$), hname$
END FOR chr
```

The types returned by HOT_TYPE are

-8	last line recall
-6	stuff keyboard queue with previous stuffer string
-4	stuff keyboard queue with current stuffer string
-2	stuff keyboard queue with defined string
0	pick SuperBASIC and stuff command
2	do code
4	execute thing
6	execute file
8	pick job
-7	not defined

Changing the HOTKEYs

Individual HOTKEYS can be turned on and off and the HOTKEY used for a particular program can changed using the HOT_OFF and HOT_SET functions. HOT_OFF and HOT_SET can return 0 (ok) or -7 (ERR.NF) if the (old) key or name cannot be found. HOT_SET can also return -9 (ERR.IU) if the new key is already in use.

```
HOT_OFF (key or program name)          turn the HOTKEY off
HOT_SET (key or program name)          ... and back on again
HOT_SET (new key, old key or name)     set new HOTKEY
```

More permanent removal of a HOTKEY is available using the HOT_REMV function. This not only turns the HOTKEY off, but removes the

definition as well. If the HOTKEY was set up using HOT_CHP, the program code and any jobs using it are removed. HOT_REMV will need to be used to remove a HOTKEY definition before re-using the particular keystroke. This is not necessary if HOT_KEY or HOT_CMD are used to re-define a string or command respectively.

```
HOT_REMV (key or program name)
```

```
ERT HOT_CHP (q, Quill, p)            Quill on ALT Q
ERT HOT_OFF (q) or ERT HOT_OFF (Quill) ALT Q turned off
ERT HOT_SET (q) or ERT HOT_SET (Quill) ALT Q back on
ERT HOT_SET (z, Quill)               Quill now on ALT Z
ERT HOT_REMV (Quill)                 Quill gone completely
```

Executing HOTKEYs Directly

There is very1ittle that is special about the HOTKEY job. A program or action set up on a HOTKEY can just as easily be invoked directly from SuperBASIC with the HOT_DO command.

```
HOT_DO key or program name
```

Additional Facilities

As the HOTKEY system provides so many ways of getting round problems with awkward software, it seems a pity not to let these fixes be used directly from the command line. The HOTKEY system includes the command EXEP to supplement the EXEC (or EX) command. This is the direct equivalent of the HOT_RES, HOT_CHP and HOT_LOAD functions. This does not set up a HOTKEY but executes a program directly.

```
EXEP file name
EXEP file name, P
EXEP file name, P , memory in kilo bytes
EXEP file name, U
EXEP file name, G
EXEP file name, G, window definition
```

HOTKEY Boot Programs

The majority of QL software falls into one of two main groups,

"resident extensions" and "transient programs". There are two other important groups, SuperBASIC programs and abominations. There is little, if any, commercial software in the form of SuperBASIC programs, and if you have written your own, then you should know how to run it! SuperBASIC programs compiled with QLiberator or Turbo are true "transient programs". Abominations should be returned to the supplier as soon as possible. If you really do need to use one, then reset your QL before and after use. The QL reset is absolute, so ritual cleansing is not required.

"Resident extensions" are provided to expand the capabilities of the QL and are designed to be loaded at the beginning of a session and remain resident in the QL for the whole of the session. The HOTKEY System is a resident extension. Other typical examples are SuperToolkit II, the Pointer Environment and the Spell extensions. Less obvious are other bits of system software such as floppy and hard disk drivers, RAM disk drivers, printer buffers and Lightning. All of these are intended to be of use for many different programs throughout an entire session.

"Transient programs" are designed to come and go as required. These are executed as required, and when you have finished with them, they go away, leaving the QLs memory free for other transient programs. Typical examples are Quill, Abacus and the other Psion programs.

Some transient programs require specific resident extensions to be present. The reasons vary. Most Qjump programs require the Pointer Environment because it makes it simple to provide the type of pop-up menus and non-destructive windows that we prefer to use. Qtyp requires the Spell extensions, because we thought that it was necessary to separate out the actual spelling checking so that it could be used in other programs as well (such as real word processors). The Editor requires the Turbo Toolkit because it is Turbo compiled SuperBASIC and uses some facilities not available in the QL ROM.

As a general rule, a BOOT file should load all the resident extensions you require, before any programs are started. This will avoid 'not complete' error messages when you try to load further extensions. The BOOT file is used in much commercial software to give users instant access to their new program - many users never progress beyond this point, but re-boot their QLs every time they wish to change programs!!!

The boundary between a supplier providing a very complex BOOT file to make it very easy to use their software, and a supplier providing so complex a BOOT file that it becomes almost impossible to use any other

supp1ier's software is a very fine one. To set up your own BOOT file, you will have to determine which resident extensions are needed for each the programs you wish to use. This should be stated in the manual, alternatively you can examine the supp1ier's own BOOT file. Any code loaded by statements of the form

```
base=RESPR(size):LBYTES mdv1_filename,base:CALL base
```

or

```
LRESPR (filename)
```
with SuperToolkit II

or

```
base=RESPR(size)
```
Loading several files into one space

```
LBYTES mdvl_filenamel, base : CALL base
LBYTES mdvl_filename2, base + a_bit: CALL base + a_bit
```
etc

may be assumed to be a resident extension. The statements can be copied into your own BOOT file at the appropriate point, and the files themselves copied onto you own BOOT disk or Microdrive. The statements may be scattered over several lines to confuse you.

Sorting out BOOT files varies from the easy (e.g. The Editor} to the impossible (CENSORED). Very easy BOOT files would consist of "EXEC mdv1_filename", in which case you need to add nothing to your own BOOT file unless you wish to HOTKEY the program with HOT_RES, HOT_CHP or HOT_LOAD. Difficult conversions are where the BOOT file indulges (and it is an indulgence} in copyright messages, pretty borders, playing tunes or other methods of obscuring the useful bits of code. Impossible BOOT files are those which include POKEs, or start a application with a CALL statement. These can sometimes be used, but require the attention of an expert machine code hacker to convert them to a sanitary form. See "abominations" above.

Some resident extensions interact with others. If this happens, then some care is required with the ordering of the resident extensions. The HOTKEY System II interacts with both the SuperToolkit II ALTKEY facility and the earlier versions of HOTKEY. For best results, load or activate SuperToolkit 1I before HOT_REXT and load your old HOTKEY file (which should be redundant} after HOT_REXT. The Pointer Environment interacts with Lightning. Load Lightning before the PTR_IMI or PTR_GEN file, and WMAN after the PTR_IMI or PTR_GEN file.

The HOTKEY system allows you to set up all your system to your own

requirements. At any time you can reconfigure your HOTKEY system by running another BASIC program with commands to change (HOT_SET}, remove (HOT_REMV} and add HOTKEYs.

In these sample BOOT files the Toolkit II LRESPR command is used. If you do not have Toolkit II (WHAT?*!?}, then you will need to use the combined statement

```
base=RESPR(space) : LBYTES name, base : CALL base
```

In these examples, the drive is specified explicitly, and the file names are between apostrophes. The first is for clarity only, the second is a personal preference.

A Simple BOOT Program (No SuperToolkit)

This sets up The Editor and QRAM on HOTKEYs. The Editor requires the resident extensions in the file "xtras". QRAM requires the PTR_GEN and WMAN extensions. The file sizes given are typical, use QRAM FILES menu or any Toolkit WSTAT command to find the actual size of each file.

```
100 REMark - Load all our extensions
110 base = RESPR(6074) : LBYTES 'flpl_xtras',base : CALL base
120 base = RESPR(9976) : LBYTES 'flpl_HOT_rext',base : CALL base
130 base = RESPR(12388): LBYTES 'flp1_ptr_gen',base : CALL base
140 base = RESPR(7762) : LBYTES 'flpl_wman',base : CALL base
150 ERT HOT_RES ('e','flpl edt_bin ,'i') : REMark The Editor
160 ERT HOT_RES ('/','flpl_qram')        : REMark Qram main program
170 HOT_GO
```

A Psion Boot Program

This boot file sets up a Psion plus Qtyp HOTKEY system. All four Psion programs are permanently resident, although only Quill is started.

```
100 REMark - Load all our extensions
110 :
120 TK2_EXT : REMark you may need this
130 LRESPR 'flpl_HOT_rext' : REMark HOTKEY extensions
140 LRESPR 'flpl_ptr_gen' : REMark the Pointer Environment
150 LRESPR 'flpl wman'
160 LRESPR 'flp1_qtyp_spell' : REMark spelling checker extensons
```

```
170 :
180 REMark - Extensions loaded, stuff our QL full of the
190 REMark - resident programs we always have available
200 :
210 REMark ERT HOT_RES ('/', 'flpl_qram') : REMark No Qram this time
220 ERT HOT_RES ('t', 'flpl_qtyp' )   : REMark Qtyp in case we use Quill
230 ERT HOT_RES ('q', 'flpl_quill')   : REMark ALT Q for a new Quill
240 ERT HOT_RES ('a', 'flpl_abacus')  : REMark ALT A for a new Abacus
250 ERT HOT_RES ('r', 'flpl_archive') : REMark ALT R for a new Archive
260 ERT HOT_RES ('e', 'flpl_easel')   : REMark ALT E for a new Easel
270 :
280 HOT_GO : REMark get HOTKEY going
290 :
300 : REMark - now we set some HOTKEYs for pickinq jobs
310 : REMark - to pretend that we are using Taskmaster
320 :
330 ERT HOT_PICK ('0', '') : REMark SuperBASIC and other no-name jobs
340 ERT HOT_PICK ('1', 'Quill')
350 ERT HOT_PICK ('2', 'Abacus')
360 ERT HOT_PICK ('3', 'Archive')
370 ERT HOT_PICK ('4' , 'Easel')
380 HOT_LIST : REMark tell us what we have please
390 PAUSE 300 : HOT_DO q : REMark start with Quill only
```

A Bigger BOOT Program

```
100 REMark - First shrink SuperBASIC's windows to leave
110 REMark - room for odd bits at the top of the screen
120 :
130 WINDOW #0;254,42,0,214   : BORDER #0;1,4,0
140 WINDOW #1;256,172,256,36 : BORDER #1;1,255
150 WINDOW #2;256,172,0,36   : BORDER #2;1,255
160 MODE 512
170 :
180 REMark - Now load all our extensions
190 :
200 TK2_EXT : REMark you may need this
210 LRESPR 'flpl_hot_rext' : REMark HOTKEY extensions
220 LRESPR 'flp1_ptr_gen' : REMark the Pointer Environment
230 LRESPR 'flpl_wman'
240 LRESPR 'flpl_qtyp_spell' : REMark spelling checker extensions
250 LRESPR 'flpl_xtras' : REMark bits and bobs for The Editor
260 :
```

```
270 REMark - Extensions loaded, stuff our QL full of the
280 REMark - resident programs we always have available
290 :
300 ERT HOT_RES ('/', 'flp1_qram')     : REMark QRAM of course
310 ERT HOT_RES ('t', 'flp1_qtyp')     : REMark Qtyp in case we use
Quill
320 ERT HOT_RES ('C', 'flp1_calc')     : REMark Pop up calculator
330 ERT HOT_RES ('k', 'flp1_calendar') : REMark ... our calendar
340 ERT HOT_RES ('W', 'flp1_alarm')    : REMark ... and the alarm
350 :
360 REMark - Now execute our permanent programs
370 :
380 FSERVE : REMark we always use the file server
390 HOT_GO : REMark get this going as well
400 EXEC 'flp1_Clock' : REMark clock around the clock
410 EXEC 'flp1_Sysmon' : REMark we need this to know what is going on
420 :
430 REMark - Now load any HOTKEYed programs that we may
440 REMark - get rid of at some time during the day
450 :
460 ERT HOT_CHP ('q', 'flpl_quill',P,32) : REMark 32K for Quill
470 ERT HOT_CHP ('a', 'flpl_abacus',P,50)
480 :
490 ERT HOT_LOAD('e', 'flpl_edt_bin','i') : REMark load The Editor
500 :
510 : REMark - now we set some HOTKEYs for picking jobs
520 : REMark - to pretend that we are usinq Taskmaster
530 :
540 ERT HOT_PICK ('0', '' ):REMark SuperBASIC and other no-name jobs
550 ERT HOT_PICK ('1', 'Quill')
560 ERT HOT_PICK ('2', 'Abacus')
570 ERT HOT_PICK ('3', 'Editor')
580 ERT HOT_PICK ('7', 'Make') : REMark we have not got to this yet
510 ERT HOT_PICK ('8', 'Clock')
520 ERT HOT_PICK ('9', 'Sysmon')
530 :
540 HOT_LIST  : REMark tell us what we have please
550 PAUSE 300 : HOT_DO e   : REMark start off with The Editor
560 PAUSE 100 : HOT_DO '0' : REMark but with SuperBASIC on top
```

HOTKEY System II V2.10

V2.10 of the HOTKEY System II introduces a number of new variations on the HOTKEY SuperBASIC extensions. These are fully functional only if you use the Pointer Interface file (PTR_GEN) version 1.26 or later.

HOT_WAKE

HOT_WAKE is a variation on the HOT_PICK function. There are two differences. The first is that after a Job is PICKed, it receives a WAKE event from the Pointer Interface: this will be ignored by most software. The second is that, if there is no Job with the right name executing when the HOTKEY is pressed, the HOTKEY System II will try to execute a THING of the same name. The HOT_RES and HOT_CHP functions set up suitable THINGs.

```
ERT HOT_RES ('/', 'Qram')      set up resident QRAM
ERT HOT_WAKE ('q', 'Qram')     set up to wake QRAM
```

If these two commands are put into your BOOT file, then pressing ALT and '/' will always EXECute a new copy of QRAM, while ALT and 'q' will PICK qram if there is a copy executing, otherwise it will EXECute a new copy.

HOT_RES1 HOT_CHP1 HOT_LOAD1

These variations on HOT_RES, HOT_CHP and HOT_LOAD are used to HOTKEY a program when you wiii normally be wanting to have at most one copy executing at a time. If there is already a program of the appropriate name executing, the HOTKEY will PICK (and WAKE) the Job. Otherwise the program will be EXECuted from the resident image (HOT_RES1 and HOT_CHP1) or LOADed and EXECuted from the default disk or Microdrive (HOT_LOAD1}.

```
ERT HOT_RES1('a', 'Abacus' ,p)
```

load Abacus into memory and set up to either EXECute it or PICK it when ALT and 'a' are pressed.

```
ERT HOT_LOAD1('e', 'QD')
```

set up to LOAD and EXECute QD the first time ALT and 'e' are pressed, otherwise WAKE it

Other WAKE Events

Versions 1.26 onwards of PTR_GEN include the facility to WAKE a Job if it is picked using the DO button or ENTER key.

HOTKEY System II - Version 2.24

The functions to set up Hotkeys to execute files or Things and the EXEP procedure have been extended to include parameter passing and to allow for programs which change their own Job names (e.g. QPAC2 Files).

You can now follow the filename or Thing name by a parameter string which will be passed to the Job when it is started. The parameter string should be preceded by a semicolon.

For those functions which incorporate a Wake Hotkey (HOT_RES1, HOT_CHP1, HOT_LOAD1 and HOT_WAKE) it is possible to specify a name for waking the Job which is different from the filename or Thing name. The effect of this differs from the Job name previously allowed, in that using this variation does not add the name to the front of the job, but merely notes that the Job will be different from the filename. The Wake name should eb given after the filename (and parameter string), preceded by an exclamation mark to distinguish it from the Job name. You should not specify both a Job name and a Wake name.

A Job name is now allowed for both HOT_THING and HOT_WAKE.

All parameters except the key and the Thing name or filename are optional.

HOT_RES	(*key,filename;params,Job name,options*)
HOT_RES1	(*key,filename;params,Job name!Wake name, options*)
HOT_CHP	(*key,filename;params,Job name,options*)
HOT_CHP1	(*key,filename;params,Job name!Wake name,options*)
HOT_LOAD	(*key,filename;params,Job name,options*)
HOT_LOAD1	(*key,filename;params,Job name!Wake name,optlons*)
HOT_THING	(*key,Thing name;params,Job name*)
HOT_WAKE	(*key,Thing name;params,Job name!Wake name*)

EXEP *filename;params,Job name,options*

Examples

The meaning of any parameters you give will depend on the application being invoked. These are not necessarily representative examples.

EXEP QD;'flpl_boot' - *start QD editing flpl_boot*
HOT_WAKE ('d',Files;'\MD'!'Delete') - *set up a Delete Hotkey*
HOT_WAKE ('D',Files;'\MD\OV\D _ERR', 'DERR') - - *called DERR*

Parameters to QPAC2 Programs

From version 1.12 of QPAC2 it is possible to pass parameters to the various menu programs. As the possibilities offered by parameter passing are very extensive, this may seem very complex. Do not worry, there is no need to do this, if you are in any doubt, please do not read the rest of this note. Advanced users of the HOTKEY System II and QPAC2 may find these facilities useful which is why they are included here.

The form of the parameter strings is a "key" followed by a value (usually in the form of a string of characters). A key is a backslash followed by a letter, followed by the value. There may be spaces between the key and the value. The keys may be upper or lower case.

Standard Parameters

`\Z xpos,ypos` `Start off the menu asleep`
This key sets up the menu as a sleeping button. If a button position is given, this should be in pixel co-ordinates from the top left corner. If the position is not given, the button will be put into the Button Frame.

`\B value` `Button Colourway (value 0 to 3)`
This button specifies the button colour for a menu set up as a sleeping button. If you use \B, you do not need \Z unless you wish to specify a button position.

`\N characters` `Button name`
This key specifies the name that will appear in the sleeping button. If you use \N, you do not need \Z unless you wish to specify a button position.

```
\C value,value     Colourways
```
This key sets the main border menu and menu window colourways.

Files Menu Parameters

```
\M command key     Menu (Copy, Move etc.)
```
The command key should be the selection key for the particular command. This is usually the first letter of the command. If so, you may give the full command name (e.g. C or COPY).

```
\O options  Options(V=View,T=Tree,S=Statistics,Z=Sleep)
```
The option letters should follow the key. If the Z option is given, then when you press ESC, the menu will go to sleep, otherwise the menu will remove itself. If you give any options, then you must give all the options you require as this overrides all defaults.

```
\S +/- order       Sort Order
```
The sort order should be given as + or - and a single letter (N for Name, T for Time etc). The + sign is optional.

```
\D name            Directory
```
This specifies the original directory. If the name starts with an underscore, it is added to the end of the data default directory.

Example Parameters For A Print Menu

```
\M PRINT \D win1__lst \O STVZ \S N
```

This will set up a Print menu, of all files ending with _lst, in all sub-direectories (\O T), showing the file length etc. (\O S), viewing each file before printing (\O V), ESC puts the menu to sleep and the files are sorted in name order.

```
\M P \D _lst \O \S N
```

This will set up a Print menu, of all files ending with _lst, in the current default (sub-)directory, without statistics and not viewing each file before printing, ESC removes the menu and the files are sorted in name order.

Example Parameters For An Execute Menu

```
\M E \D flp1_ \S N
```

This will set up and Execute menu for files on FLP1_, sorted in name order. The default options for Statistics etc will be used.

Example Parameters For A View Menu

```
\C 1,1 \O V \S -T
```

This will set up a (black and red) View menu, of files in the current default (sub-)directory, without statistics and sorted with the most recently updated file first. ESC removes the menu.

Using The HOTKEY System II To Set Up QPAC2 Parameters

The HOTKEY System II (v2.21 onwards) can be used to pass parameters to the QPAC2 menus. The parameters should be in a string (or string variable) after a semicolon.

```
ERT HOT_THING('p','Files';'\M P \D win1__lst \O STV \S N')
```

This sets up a Hotkey to Execute the Files menu and pass it a parameter string to set up the Files menu as Print menu. You need to be a little more careful if you are going to set up a Wake Hotkey, because the Files Menu changes its own Job name when an operation (such as Print) is selected. So in order to Wake the menu, you need to specify a Wake name after an exclamation mark. (Note that spaces are not very important!)

```
ERT HOT_WAKE
('p','Files';'\MP\Dwin1__lst\OSTV\SN'!'Print')
```

This will try to Wake a "Print" Job. If this fails, it will Execute the "Files" Thing with the parameters to set up a "Print" Job.

```
EXEP 'Files';'\B3 \MD \OV \S-T'
```

This will set up a Black and Green button to View and Delete files fromthe current (sub-)directory, with the most recently updated files first in the list.