QMENU

How to Program and Use
The Menu Extension

Revision 8

Jochen Merz Software
Kaiser-Wilhelm-Str. 302
47169 Duisburg
Germany

Web: SMSQ.J-M-S.COM

The Menu Extension

This software package offers the user a truly simple to use interface, which, instead of continually confronting one with overlays and menus, offers easy to use, practical menus. Not only the machine-code programmer but also the Super BASIC user is now able to take comfortable advantage of simple menus. Whether it's for a short program for your own use or for a commercially viable project - this is the one for you.

The original version, written over five years ago, has been re-written and improved several times to give the latest version available today. At this point I would like to thank Tony Tebby for his valuable support and many helpful ideas. The Menu Extension is actually still not the final product. The SuperBASIC interface, for example, does not allow multiple return parameters. Other features will be implemented as soon as time allows. As you can see, the Menu Extension is growing all the time! New menus are being added, others are improved.

But let's get down to the nitty-gritty of its capabilities, so that it can be used by other programmers. It’s important for me that the Menu Extension gets used, gets talked about - gets bought! That’s the reason for the low price of this program package. But because every new and important change of the program requires new documentation, there will be a small update charge. As far as commercial applications go, of course all copyrights of the program and the documentation remain with Jochen Merz. A passing on or copying of the program without the written permission of Jochen Merz is forbidden. For commercial purposes, i.e. using the Menu Extension as part of marketed software packages, a copying of the program is only permitted on payment of a licensing fee of 1.50 EUR net per copy. The software does not have to be linked to compiled software, it can be delivered as a separate file on the storage medium. No other file on the storage medium delivered with this software may be copied. That part of the documentation pertaining to the user side of the Menu may also be given to the customer. This does not include the programming guide for machine code or SuperBASIC, nor the menus which are not used! The basic condition for all this is the written permission of Jochen Merz Software. Please ask for further information.

If you just take the FILE-SELECT function as an example, you'll soon see what a powerful support system Menu Extension can be. FILE-SELECT is one of the most complex menu of the Extension, not only because loading files is a part of almost every computer activity. And above all today, when hard disks, sub-directories etc. running under QDOS, are (thank heavens!) pretty much an everyday thing, we need a capable file selecting system more than ever. Look at one of the examples. You'll see that there's simply no better way.

Machine Code Interface

A menu is called by using the Menus-Thing. This Thing is an extension Thing, i.e. it contains a linked list of various routines which offer a variety of pre-defined but flexible menus.

The use of a menu normally follows five steps:
- Prepare call-up parameter
- Use Menus-Thing
- Call the menu subroutines
- Release Menu-Thing
- Process the return parameter

At present, there are following menus, each identified by a four capital letter abbreviation ID.

FSEL	File Select
DSEL	Directory Select
XSEL	Extension Select
ITSL	Item Select
RPER	Report Error
FERR	File Error
RSTR	Read String
LIST	List Selection (very general)
VIEW	View File
BUTN	Button
DORP	Do and Report
CHSL	Character Select
INFO	Read default settings
SET	Set the default settings
XBTN	Extended feature Button
SYSS	Select from system lists

The calls are carried out with the same format. The ID abbreviation is loaded into D2. A1 is a pointer to the parameter table. It comprises of a number of long words, with contain the parameter or the pointer to the parameter. Behind every parameter is the meaning, followed by the type in the next line. Parameters may be of several types, which are defined as follows:

```
thp.char        equ     $0004   character
thp.ubyt        equ     $0008   unsigned byte
thp.sbyt        equ     $000a   signed byte
thp.uwrd        equ     $0010   unsigned word
thp.swrd        equ     $0012   signed word
thp.ulng        equ     $0020   unsigned long
thp.slng        equ     $0022   signed long
thp.fp8         equ     $0042   eight byte floating point
thp.str         equ     $0100   string (pointer)
thp.sstr        equ     $0200   sub-string (pointer)
thp.nnul        equ     $0800   negative null (-1) (no thp.ptr)
thp.arr         equ     $0800   array (thp.ptr)
thp.opt         equ     $1000   optional
thp.upd         equ     $e100   updated parameter
thp.call        equ     $c000   call parameter
thp.ret         equ     $a000   return parameter
thp.ptr         equ     $8000   call or return parameter
```

Every parameter uses one or two longwords in the parameter table. For simple values, it is just the value, right justified in the longword. For pointer types, the most significant word of the first longword contains the type, the less significant word contains additional information. The second longword is the pointer to the value, which have to be left justified starting from the supplied address.
The additional information is the maximum string length for return strings, or, in case of an array, the array type.

Arrays are defined as follows: the pointer points to an array descriptor block of the following form:

long	pointer to array contents
	if array-type is 0, pointer is absolute, if 1, long relative
word	number of dimensions

for every dimension

word	number of elements
word	element-multiplier

The array contents is a regular table of elements.

The enclosed library contains two sub-routines for using and freeing the Menu Extension. Look for user examples in the Assembler Sources of the example programs. The entry jumps for both routines are defined as follows:

```
Use Menu Extension.                          UT_USMEN

      Entry                                  Exit
D2.l  Extension ID
D3.l                                         Version
A1                                           Address of Thing

Error returns:
      ERR.NI   THING not implemented
      ERR.NF   Menus not found
      any returns from Menu
```

```
Free Menu Extension.                         UT_FRMEN

Error codes are preserved
```

```
Standard Menu Action Routine

      Entry                                  Exit
D0                                           0, +1 or error
A1    ptr to parameter list                  preserved

All registers except D0 preserved
```

All menus apart from BUTN require a primary window with outline. Because the programs which call a menu already have a main window managed by Window Manager, this should pose no problem.

The programmer may pass -1 as x- and y-position to every menu, which then positions the window at the current pointer position.

For compatibility reasons, the programmer should pass a value of -1 (or 0 to 3) for the colourways. The colourways are not required for SMSQ/E, as the Menu Extension uses the Colour Schemes of SMSQ/E's HiColour drivers. The parameter fields only exist for downward compatibility and are ignored – except for new timeout functionality in some menus and may be used in future versions of the Menu Extension for additional features.

FILE-SELECT (FSEL)

The FILE SELECT function can be given a Menu name (title), a default file name, the name of a directory key (0=no directory, -1=DATA default, -2=PROG default), the endings of file names, size, position and menu attributes. The routine works on a copy of the default file name. When a file name is selected, D0 is returned with 0 and the file name is copied into the filename return buffer. If the user presses ESC, D0 is set to +1. If D0 is not 0, then the filename return buffer is not changed. The filename return buffer should be at least 38 bytes long.
If the default filename is an '_', all items except the directory select item are set to unavailable. This allows the programme to make sure that only existing files are selected, or even only existing files with a given extension. If the default filename is just a single space " ", then the current hotkey stuffer buffer content is used instead.
The extension may be set unavailable by adding an '_' to the default ending. All items except directory can be disabled by passing a suggestion of '_'.
If the update bit in the directory or extension parameter is set, then, if the user changes the extension or the directory, the string is updated (if it is not a key).
If the DEV device is installed, 'DEV' is not displayed in the drivers list. If names or directories starting with DEV are passed, they are substituted with the device defined to the given DEV.
The minimum window size of the outlined primary window has to be 444x98 pixels.

The format of the parameter list:

fs_mennm	equ	$04	pointer to menu name (or 0)
	thp.call+thp.opt+thp.str		
fs_defnm	equ	$0c	pointer to default name (or 0)
	thp.call+thp.opt+thp.str		
fs_dirtp	equ	$10	directory type (if long must be 0, -1 or -2)
	thp.upd+thp.opt+thp.str+thp.slng		
fs_dirnm	equ	$14	pointer to directory name or key, the pointer itself could be 0, -1 or -2
fs_extnm	equ	$1c	pointer to extension name (or 0)
	thp.upd+thp.opt+thp.str		
fs_size	equ	$20	lower byte: number of lines (or 0)
	thp.opt+thp.uwrd		
fs_xpos	equ	$24	x position (or 0) of left hand side
	thp.opt+thp.nnul+thp.uwrd		
fs_ypos	equ	$28	y position (or 0) of top
	thp.opt+thp.nnul+thp.uwrd		
fs_mainc	equ	$2c	main colourway (-1), ignored
	thp.opt+thp.ulng		
fs_filec	equ	$30	file colourway (-1), ignored
	thp.opt+thp.nnul+thp.ulng		
fs_fname	equ	$38	pointer to filename return
	thp.ret+thp.str		

DIRECTORY SELECT (DSEL)

The Directory Select function can be given a menu name, size, position and menu attributes. If a directory name is selected, D0 is passed back with 0 and the name is returned to the directory name buffer. If the user presses ESC, a+1 is returned to D0. If D0 is not 0, the return directory name buffer is not changed. The return buffer should be at least 38 bytes long. The address to which the return filename pointer points is checked for a valid directory name. If the test passed OK, it is used as a suggestion, otherwise the DATA default is used.

The format of the parameter list:

```
ds_mennm    equ     $04     pointer to menu name (or 0)
            thp.call+thp.opt+thp.str
ds_lines    equ     $08     number of lines (or 0)
            thp.opt+thp.uwrd
ds_xpos     equ     $0c     x position (or 0) of left hand side
            thp.opt+thp.nnul+thp.uwrd
ds_ypos     equ     $10     y position (or 0) of top
            thp.opt+thp.nnul+thp.uwrd
ds_mainc    equ     $14     main colourway (-1), ignored
            thp.opt+thp.ulng
ds_ddirc    equ     $18     default directories colourway (-1), ignored
            thp.opt+thp.nnul+thp.ulng
ds_fname    equ     $20     pointer to filename return
            thp.ret+thp.str
```

EXTENSION SELECT (XSEL)

The Extension Select function can be given a position and menu attributes. If an extension is selected, D0 is returned with 0 and the extension name is copied into the return buffer. If the user presses ESC, D0 is given the value +1. The return buffer should be at least 6 bytes long.

The format of the parameter list:

```
xs_xpos     equ     $00     x position (or 0) of left hand side
            thp.opt+thp.nnul+thp.uwrd
xs_ypos     equ     $04     y position (or 0) of top
            thp.opt+thp.nnul+thp.uwrd
xs_mainc    equ     $08     main colourway (-1), ignored
            thp.opt+thp.ulng
xs_fname    equ     $10     pointer to filename return
            thp.ret+thp.str
```

ITEM SELECT (ITSL)

The Item Select function can be given a menu name, a request, up to three menu options, position and menu attributes. The request can be longer than one line. The text may contain CHR$(10) (linefeeds) or a backslash (will be replaced by CHR$(10)). Please make sure that ROM code contains only CHR$(10) and no backslashes! The first menu option appears approximately in the centre of a window, a further, second option to its right and any third menu option to the left. The pointer is always above the first menu option. The first letter of every menu option is also used for the keyboard selector. If a menu option comprises of a string containing only CHR$(27) (Escape), this string will be replaced by "ESC". The keyboard selection is also done with the ESC key.
A timeout can be provided in the MSW of is_mainc. The window will then disappear after timeout, even if the user has not selected anything.
The function returns one of five possible values.

0	DO or ESC in window, but no option has been selected
1	Middle option selected
2	Right hand option selected
3	Left hand option selected
-3	Timeout – nothing selected

If the user presses ESC when no ESC option exists, D0 is returned with +1.
The pointer may point to a longword instead of a string which should contain a standard error code, which is then expanded into text and displayed.

The format of the parameter list:

```
is_mennm    equ     $04     pointer to menu name (or 0)
            thp.call+thp.opt+thp.str+thp.slng
is_prmpt    equ     $0c     pointer to prompt (or 0)
            thp.call+thp.opt+thp.str+thp.slng
is_item1    equ     $14     pointer to text of item 1
            thp.call+thp.str
is_item2    equ     $1c     pointer to text of item 2 (or 0)
            thp.call+thp.opt+thp.str
is_item3    equ     $24     pointer to text of item 3 (or 0)
            thp.call+thp.opt+thp.str
is_xpos     equ     $28     x position (or 0) of left hand side
            thp.opt+thp.nnul+thp.uwrd
is_ypos     equ     $2c     y position (or 0) of top
            thp.opt+thp.nnul+thp.uwrd
is_mainc    equ     $30     optional timeout in upper word, else -1
            thp.opt+thp.ulng
is_itnum    equ     $38     item number return
            thp.ret+thp.ulng
```

REPORT ERROR (RPER)

The Report Error procedure can be given an error code, position and menu attributes. If the user presses ESC, D0 is returned with +1.
The error code may be zero, in which case the message 'no error' appears. The special errors (e.g. pointer to string with bit 31 set) are also reported.
For long error messages, the window is dynamically widened.
A timeout can be provided in the MSW of er_mainc. The window will then disappear after timeout, even if the user has not acknowledged it by clicking.

The format of the parameter list:

```
er_error    equ     $00     error code
            thp.ulng
er_xpos     equ     $04     x position (or -1) of left hand side
            thp.opt+thp.nnul+thp.uwrd
er_ypos     equ     $08     y position (or -1) of top
            thp.opt+thp.nnul+thp.uwrd
er_mainc    equ     $0c     optional timeout in upper word, else -1
            thp.opt+thp.ulng
```

FILE ERROR (FERR)

This is a standard file-error handler. All errors will be reported with explanatory text, and the user has the choice of selecting retry or abort. On some errors, he may also select overwrite or the choice to edit the name. The return parameter is -1 if no file-error has been passed, 0 for abort or ESC, 1 for retry, 2 for overwrite and 3 for edit.
The following errors are file-errors: -1, -2, -4 bis -13, -15, -16, -20, -22.
If option is non-zero, overwrite and edit will be presented if necessary.

The format of the parameter list:

```
fe_error    equ     $00     error code
            thp.ulng
fe_optns    equ     $04     options allowed
            thp.opt+thp.ubyt
fe_xpos     equ     $08     x position (or -1) of left hand side
            thp.opt+thp.nnul+thp.uwrd
fe_ypos     equ     $0c     y position (or -1) of top
            thp.opt+thp.nnul+thp.uwrd
fe_mainc    equ     $10     main colourway (-1, 0 to 3)
            thp.opt+thp.ulng
fe_actn     equ     $18     selected action
            thp.ret+thp.slng
```

READ STRING (RSTR)

The Read String function can be given a menu name, default name, request text, size, position and menu attributes. The request can be longer than one line. The text may contain CHR$(10) (linefeeds) or a backslash (will be replaced by CHR$(10)). Please make sure that ROM code contains only CHR$(10) and no backslashes! The routine works with a copy of the default name. If a name is entered, D0 is passed back with 0 and the name is returned to the return buffer. If the user presses ESC, D0 is returned with +1. If D0 is not 0, the return buffer is not changed. The return buffer should be at least 38 bytes long (if no size is given), but long enough for the longest possible string. The cursor is positioned at the end of the string, unless the top word of rs_chars is set to 1 (bit 0 of the MSW set), in which case it is positioned at the start of the string. Other features can be selected by setting other bits in the MSW: setting bit 2 will force the user to enter a value, an empty string return is not allowed. Setting bit 4 will accept integers only. Setting bit 5 will accept fractions and integers. If bit 6 is set, a negative sign is allowed too. Setting bit 7 enables password entry (only asterisks are shown, and the cursor cannot be moved). Setting bit 14 disables the function of the [ESC] key in this menu.

The format of the parameter list:

```
rs_mennm    equ     $04     pointer to menu name (or 0)
            thp.call+thp.opt+thp.str
rs_defnm    equ     $0c     pointer to default name (or 0)
            thp.call+thp.opt+thp.str
rs_prmpt    equ     $14     pointer to prompt (or 0)
            thp.call+thp.opt+thp.str
rs_chars    equ     $18     LSW: nr of characters (0 defaults to 34),
                            MSW: option-bits
            thp.opt+thp.uwrd
rs_xpos     equ     $1c     x position (or 0) of left hand side
            thp.opt+thp.nnul+thp.uwrd
rs_ypos     equ     $20     y position (or 0) of top
            thp.opt+thp.nnul+thp.uwrd
rs_mainc    equ     $24     main colourway (-1), ignored
            thp.opt+thp.ulng
rs_fname    equ     $2c     pointer to filename return
            thp.ret+thp.str
```

VIEW FILE (VIEW)

This is the View File procedure. It is given the name of the file to be viewed, an optional "start string", an optional "end string", the number of character per line and menu attributes. The routine attempts to open a window as large as possible. If the routine is unable to open the file, an error window is overlaid. If the start and end strings are given, only that part of the file inside these two strings is displayed. If nothing is given then the entire file can be viewed. Minimum number of characters is 16.

The format of the parameter list:

```
vw_filnm    equ   $04    pointer to file name
            thp.call+thp.str
vw_start    equ   $0c    pointer to start string (or 0)
            thp.call+thp.opt+thp.str
vw_end      equ   $14    pointer to end string (or 0)
            thp.call+thp.opt+thp.str
vw_chars    equ   $18    number of characters (or 0 defaults to 80)
            thp.opt+thp.uwrd
vw_mainc    equ   $1c    main colourway (-1), ignored
            thp.opt+thp.ulng
vw_listc    equ   $20    list colourway (-1), ignored
            thp.opt+thp.nnul+thp.ulng
```

BUTTON (BUTN)

The Button procedure must be given a button name or a pointer to a sprite. The sprite must not be larger than 40x20 pixels. Position and menu attributes can also be given. The job which calls this procedure may not have any windows open, otherwise the button cannot be positioned correctly. It first tries to place the button in QPAC II's Button Frame. If it is unable to do this, the position given is used.

The format of the parameter list:

```
bt_name     equ   $04    pointer to button name or to sprite
            thp.call+thp.str+thp.ulng
bt_xpos     equ   $08    x position (or -1) of left hand side
            thp.opt+thp.nnul+thp.uwrd
bt_ypos     equ   $0c    y position (or -1) of top
            thp.opt+thp.nnul+thp.uwrd
bt_mainc    equ   $10    main colourway (-1), ignored
            thp.opt+thp.ulng
```

EXTENDED BUTTON (XBTN)

The Extended Button function must be given a button name or a pointer to a sprite. The sprite must not be larger than 40x20 pixels. Position and menu attributes can also be given.
Optionally, there may be pointer to a second sprite. Instead of a pointer, the value +1 print a WAKE symbol, the value +2 prints a "Changed but not saved" symbol.
In the option control word set bits define:

Bit 0	second sprite is separate item, which is selected with keystroke WAKE.
Bit 8	second sprite should flash (WMAN 1.52 or higher).
Bit 16	HIT is valid selection keystroke (Button immoveable).
Bit 17	ESC is valid selection keystroke.

The routine returns the keystroke, which woke the button (0=ESC, 1=WAKE, 2=HIT, 3=DO). The job which calls this procedure may not have any windows open, otherwise the button cannot be positioned correctly. It first tries to place the button in QPAC II's Button Frame. If it is unable to do this, the position given is used.

The format of the parameter list:

```
bt_name     equ   $04     ; pointer to button name
            thp.call+thp.str+thp.ulng
bt_xpos     equ   $08     ; x position (or -1) of left hand side
            thp.opt+thp.nnul+thp.uwrd
bt_ypos     equ   $0c     ; y position (or -1) of top
            thp.opt+thp.nnul+thp.uwrd
bt_mainc    equ   $10     ; main colourway (-1), ignored
            thp.opt+thp.nnul+thp.ulng
bt_spr2     equ   $14     ; ptr to second optional sprite (or 1, 2)
            thp.opt+thp.ulng
bt_opts     equ   $18     ; option control word
            thp.opt+thp.ulng
bt_retp     equ   $20     ; button selection keystroke:
            thp.ret+thp.ulng
```

LIST SELECT (LIST)

List Select may be called for two different variations: one menu item may be selected and the routine returns, and all items are available for selections. It makes no difference whether the item is selected by hit or do. The return parameter is the menu item number (counting starts at 0), or -1 for ESC or -2 for OK.
The second variation needs a status list, which contains the status of every item in the list. The user may select or deselect as many items as required, and the routine returns on OK or do only. The status list is returned with the updated item statuses, and the return parameter contains the same values as on the first variation.
The menu item list is passed as a 2-dimensional string array.
The status list is passed as a 1-dimensional integer array. If the status list has not the same number of items, it is ignored. The less significant byte in the word contains the status of the corresponding item: WSI.AVBL, WSI.UNAV or WSI.SLCT (which is $00, $10 and $80). The most significant byte is used for controlling whether the item can be edited. At the moment, only bit 6 is used to signal that the item might be edited if set. Bit 0 is set on return if the item has been edited. The other bits of the most significant byte should be clear.
The bits in the menu-item-control-longword have the following meaning, if set:

Bit 0	the first character will be the selection keystroke, it is underscored.
Bit 2	selection keystrokes will be placed in front of the items.
Bit 4	the menu items will be centred.
Bit 6	treat a [Hit] as a [Do]

Bits 0 and 2 are exclusive. All unused bits have to be set to zero!
The number of lines and columns is the maximum, so less lines or columns may be used. The routine optimises the string length, if possible.

The format of the parameter list:

```
ls_mennm    equ     $04     ; pointer to menu name (or 0)
            thp.call+thp.opt+thp.str
ls_itlty    equ     $08     ; item list type
            thp.call+thp.arr+thp.str
ls_itlst    equ     $0c     ; item list
ls_stlty    equ     $10     ; status list type
            thp.upd+thp.opt+thp.arr+thp.uwrd
ls_stlst    equ     $14     ; item status list (or 0)
ls_mictl    equ     $18     ; menu-item-control
            thp.opt+thp.ulng
ls_lines    equ     $1c     ; max number of lines (or 0)
            thp.opt+thp.uwrd
ls_colms    equ     $20     ; max number of columns (or 0)
            thp.opt+thp.uwrd
ls_xpos     equ     $24     ; x position (or 0) of left hand side
            thp.opt+thp.nnul+thp.uwrd
```

```
ls_ypos     equ   $28     ; y position (or 0) of top
            thp.opt+thp.nnul+thp.uwrd
ls_mainc    equ   $2c     ; main colourway (-1), ignored
            thp.opt+thp.ulng
ls_listc    equ   $30     ; list window colourways (-1), ignored
            thp.opt+thp.nnul+thp.unlg
ls_item     equ   $38     ; pointer to return item number
            thp.ret+thp.uwrd
```

Do and Report (DORP)

Do and Report can be used for various reasons: either to report the progress of an action and/or to report a final result. The menu can disappear automatically after the action has been done, or wait for an OK, or wait for an optional "Again", or, in error case, could allow "Abort" and "Retry". A good example for this menu would be a menu which allows formatting a disk.

This menu needs a menu title, a textual description of the action, an optional position and optional menu attributes, optional size specifier and, of course, the address of the action routine.
This routine sets up a window, writes out what is going to happen, does the action and reports the result from this action (e.g. format, makedir etc.). It is also possible to report progress of the action.
If the user escapes (i.e. aborts), d0 is returned = +1.
The menu control bits define the following features (if bit set):

Bit 0	do not wait for OK (just return)
Bit 2	no error: allow Again; error: allow Retry and change OK to Abort

The format of the parameter list:

```
do_mennm    equ   $04     ; pointer to menu name (or 0)
            thp.call+thp.opt+thp.str
do_actds    equ   $0c     ; pointer to textual action description (or 0)
            thp.call+thp.opt+thp.str
do_chars    equ   $10     ; number of characters for result string (default
                          is 40)
            thp.opt+thp.uwrd
do_xpos     equ   $14     ; x position (or -1) of left hand side
            thp.opt+thp.nnul+thp.uwrd
do_ypos     equ   $18     ; y position (or -1) of top
            thp.opt+thp.nnul+thp.uwrd
do_mainc    equ   $1c     ; main colourway (-1), ignored
            thp.opt+thp.ulng
do_mictl    equ   $20     ; menu item control
            thp.opt+thp.ulng
do_pact     equ   $24     ; pointer to action routine
            thp.ulng
```

The action routine is called after the window has been drawn; it should perform the action and return either an error or a result string. As a special feature, the window is set to the result window, so that progress reports can be printed, either directly (with short timeout) or by using WM.STIOB first and then WM.IDRAW. The entry/exit parameters for the action routine are:

	Entry	Exit
D0.l	0 for first call, else set	error or 0
D1.l	info window/object nr.	smashed
D3.w	maximum result string length	smashed
A0	window channel ID	preserved
A1	ptr to window in wdef	result string (if d0=0)
A2	window manager vector	preserved
A4	window working area	preserved

System-List Select (SYSS)

This is an easy method of selecting from lists which show system-relevant information.
This menu needs a menu title, and a contents list type. At present, several lists are available:
Hotkey stuffer buffer history (content type 0): all the strings, filenames etc. which have been placed into the stuffer buffer history). Please note that as long as there is no or just a current stuffer buffer entry available, no list will appear. As soon as two or more entries exist, the list will show all of them. There is also the possibility of getting lists of all currently running jobs (content type 16), all things available in the whole system (content type 32), and a list of executable things only (content type 34).
The return parameter is the selected item, e.g. stuffer buffer string, job-name, thing-name.
Optional position and optional size control can also be specified.
The bits in the menu-item-control-longword have the following meaning, if set:

Bit 0 the first character will be the selection keystroke, it is underscored.
Bit 2 selection keystrokes will be placed in front of the items.
Bit 4 the menu items will be centred.

Bits 0 and 2 are exclusive. All unused bits have to be set to zero!
The number of lines and columns is the maximum, so less lines or columns may be used. The routine optimises the string length, if possible.

The format of the parameter list:

```
ss_mennm    equ     $04     ; pointer to menu name (or 0)
            thp.call+thp.opt+thp.str
ss_ctype    equ     $08     ; list contents type
            thp.opt+thp.ulng
ss_mictl    equ     $0c     ; menu-item-control
            thp.opt+thp.ulng
ss_lines    equ     $10     ; max number of lines (or 0)
            thp.opt+thp.uwrd
ss_colms    equ     $14     ; max number of columns (or 0)
            thp.opt+thp.uwrd
ss_xpos     equ     $18     ; x position (or 0) of left hand side
            thp.opt+thp.nnul+thp.uwrd
ss_ypos     equ     $1c     ; y position (or 0) of top
            thp.opt+thp.nnul+thp.uwrd
ss_mainc    equ     $20     ; main colourway (-1), ignored
            thp.opt+thp.ulng
ss_listc    equ     $24     ; list window colourways (-1), ignored
            thp.opt+thp.nnul+thp.unlg
ss_str      equ     $2c     ; pointer to return string
            thp.ret+thp.uwrd
```

Character Select (CHSL)

Character select allows the user to select a charcter out of a list of characters or the whole character set.
It needs an optional menu name, an option word, optional font for the character table sub-window, optional position and optional colour attributes.
The return string may contain a character on call, which is used as default "selected" character. The return string should be set to either 0 length or 1, followed by the character. No other length should be given for future compatibility reasons.
The current version will optionally position the pointer over a character. Bit 15 in the option bits has to be set and the most significant byte of this longword (Bits 31..24) should contain the ASCII value of this character.
If the user escapes via ESC, the return value is a string with zero length and D0 is set to +1. If the user leaves the menu with OK, the return string is set to the selected character.
The availability bits are defined as follows (bits set for feature):

Bit 0	non-printable characters
Bit 1	digits
Bit 2	lower-case characters
Bit 3	upper-case characters
Bit 4	remaining printable characters
Bit 6	cursor control characters
Bit 15	position pointer over character given in MSB.

The format of the parameter list:

```
ch_mennm    equ     $04     ; pointer to menu name (or 0)
            thp.call+thp.opt+thp.str
ch_avbit    equ     $08     ; availability bits
            thp.opt+thp.ulng
ch_avlbl    equ     $10     ; availability table (not implemented)
            thp.call+thp.opt+thp.arr+thp.uwrd
ch_font     equ     $14     ; font used for subwindow
            thp.opt+thp.ulng
ch_xpos     equ     $18     ; x position (or -1) of left hand side
            thp.opt+thp.nnul+thp.uwrd
ch_ypos     equ     $1c     ; y position (or -1) of top
            thp.opt+thp.nnul+thp.uwrd
ch_mainc    equ     $20     ; main colourway (-1), ignored
            thp.opt+thp.ulng
ch_listc    equ     $24     ; table window colourways (-1), ignored
            thp.opt+thp.nnul+thp.ulng
ch_retst    equ     $2c     ; pointer to return string
            thp.ret+thp.str
```

Get default settings (INFO)

This function allows the user to read a default setting which is configured within the Menu Extension. At present, all eight default file extensions, all eight default directories can be read as well as the pre-defined colours.
On entry to the call, an inquiry key is passed. The returned string is the result.

The values for the inquiry keys currently defined are:

0 .. 7	default extension 1 to 8 (6)
32 .. 39	default directory 1 to 8 (38)
64	default main colour (unused in current version)
65	default sub-window colour (unused in current version)
66	default button colour (unused in current version)
96	default update-icon flash frequency (2)
97	default delay for “hint” windows (2)

The minimum size for the return string is given in brackets.

The format of the parameter list:

```
inf_key     equ     $00     ; inquiry key
            thp.ulng
inf_ret     equ     $08     ; return string
            thp.ret+thp.str
```

Set default settings (SET)

This procedure allows the user to set a default, which can be read by other programs using the INFO call (see above). The values for the inquiry keys are exactly the same.

The format of the parameter list:

```
set_key     equ     $00     ; inquiry key
            thp.ulng
set_val     equ     $08     ; pointer to value string
            thp.call+thp.str
```

Files on the disk

The QMENU disk contains a lot of files. You will find several examples in machine code and BASIC, useful key files for assembler programming, useful utility routines and much more.

You can use the files to produce your own calls to Menu routines. You can link them in if you find them useful.

The HELP files can be used together with the QD Help system. Just copy all the files into the help system location and refer to the QD manual.

We have also put the “usermanual_txt” (printed further down in this manual under “The User Side”) as a plain txt file onto the disk – you can take all of this or relevant parts of it into your manual, if you produce a software product which uses the Menu Extension.
Please contact J-M-S first for a license agreement – shipping the Menu Extension together with your product is not expensive – just EUR 1.50 per copy.

SuperBASIC-Interface

Entry is by procedures and functions in the normal manner. Optional parameters, which are not required, are simply left out. If other parameters are to follow the omitted ones, then a comma must be given for every parameter omitted. Some examples can be seen below.

All menus except BUTTON_WAIT take for granted that the primary window (#0 in SuperBASIC) has been outlined using the OUTLN command.

Please refer to the SMSQ/E manual if you need information about OUTLN.

FILE_SELECT$ ([menu_name] , [suggestion] , [directory] , [extension] , [size] , [x_pos , y_pos])

This function calls up a comfortable filename selection window.
If a DEV-device exists in your system, 'DEV' is not shown. Filenames passed which start with DEV are automatically expanded, as well as DATA- or PROG-Default, if they point to DEV.

The parameters are defined as follows:

menu_name: Title for the window.

suggestion: A filename is given for editing, which is then accepted with OK. If the suggestion is an '_', all items except the directory select item are set to unavailable. This allows the programme to make sure that only existing files are selected, or even only existing files with a given extension. If the suggestion filename is just a single space " ", then the current hotkey stuffer buffer content is used instead.

directory: A listing of this directory is shown. Names such as 'flp1_exec_' or 'ram1_' can be given here. Enter -1 for the current DATA default, or -2 for the current PROG default.

extension: For specified file endings. The ending should be no more than four characters long, e.g. _bas or _ASM. If a directory is specified, this will show only those files with this ending. Subdirectory names excepted! If the ending ends in an '_', this item is set to unavailable. Instead of an extension, a filetype can be given (0, 1 or 2).

size: If this parameter is omitted, the window will be drawn as large as possible. Here you can specify the number of lines in a file name window (if scrollable, you lose one line for the two arrow-blocks). To force a one-column window, add +256.

x_pos,y_pos: These define the position of the window. If these parameters are omitted, the current pointer position is used.

Examples:

```
LOAD FILE_SELECT$('Start BASIC program',,win1_basic_,_bas)
file$=FILE_SELECT$(,-1,,,7,,)
```

DIR_SELECT$ ([menu_name] , [lines] , [x_pos , y_pos] ,[-1],[-1], [default_dir])

This function is used to select devices or subdirectories. Up to 8 directories can be pre-set by the user (using CONFIG).

The parameters are defined as follows:

menu_name: Title for the window.

lines: If this parameter is omitted, the window will be drawn as large as possible. Here you can specify the number of lines in the directory window (if scrollable, you lose one line for the two arrow-blocks). Adjust the window size here.

x_pos,y_pos: These define the position of the window. If these parameters are omitted, the current pointer position is used.

default_dir: is the default directory which is given in the F2-item; here starts the directory listing.

Examples:

```
PRINT DIR_SELECT$
directory$=DIR_SELECT$('Select drive',,0,0)
```

EXT_SELECT$ ([x_pos , y_pos])

This function is used to select a pre-defined file ending.

The parameters are defined as follows:

x_pos,y_pos: These define the position of the window. If these parameters are omitted, the current pointer position is used.

Example:

```
PRINT EXT_SELECT$
```

ITEM_SELECT ([menu_name] , [request] , option1 , [option2] , [option3] , [x_pos , y_pos] , [timeout])

This function enables the user to make a selection of one to three menu options. The size of the window is automatically calculated in order that all options and any request have enough room. The first letter of every menu option is also used for keyboard selection. If a menu option is a string containing only CHR$(27) (Escape), this string will be replaced by "ESC". The ESC key is used for keyboard selection.
If the user ESCapes, the return value is 0. Otherwise it is 1 for option 1, 2 for option 2, 3 for option 3. If a timeout has been given and nothing has been selected, the return value is -3.

The parameters are defined as follows:

menu_name: Title for the window. Can be a string, or an error code (which is automatically converted into the error string).

request: Explanatory test. It can be longer than one line, may include CHR$(10) (linefeeds) or a backslash, which will be replaced by CHR$(10). Please make sure that the ROM-code contains only CHR$(10) and no backslashes!

option1: The first menu option, shown approximately in the centre of the window.

option2: The second menu option, shown to the right of the first.

option3: The third menu option, shown to the left of the first.

x_pos,y_pos: These define the position of the window. If these parameters are omitted, the current pointer position is used.

timeout: An optional timeout can be given: multiply the number of ticks (1/50th of a second) by 65536.

Examples:

```
PRINT ITEM_SELECT('Big choice',"Select any old thing\(even
      though it won't do anything)",'Wowee!',great,super)
PRINT ITEM_SELECT(-8,'Overwrite?',Yes,No)
PRINT ITEM_SELECT(,,more,CHR$(27))
```

REPORT_ERROR errorcode , [x_pos , y_pos] , [timeout]

This procedure reports an error and waits for the user to confirm this by a keypress. If a timeout is given, the window disappears after this time even without a keypress by the user.

The parameters are defined as follows:

errorcode: Standard QDOS errors (as in REPORT).
x_pos,y_pos: These define the position of the window. If these parameters are omitted, the current pointer position is used.
timeout: An optional timeout can be given: multiply the number of ticks (1/50th of a second) by 65536.

Example:
```
REPORT_ERROR -7
```

FILE_ERROR (errorcode , [option] , [x_pos , y_pos])

This function reports a file-error and waits that the user selects between "Retry" or "Abort". Optional, "Edit" or "Overwrite" may be offered for selection, depending on the type of error. The return parameter is -1 if the error is not a file system error, 0 for Abort (or ESC has been pressed), 1 for retry, 2 for overwrite and 3 for edit.

The parameters are defined as follows:

errorcode: Standard QDOS errors (as in REPORT).
option: If non-zero, some errors allow "overwrite" or "edit".
x_pos,y_pos: These define the position of the window. If these parameters are omitted, the current pointer position is used.

READ_STRING$ ([menu_name] , [suggestion] , [request] , [length] , [x_pos , y_pos])

This function reads in a string, normally that of a filename.

The parameters are defined as follows:

menu_name: Title for the window.

suggestion: A string is given for editing, which is then accepted with ENTER.

request: An explanatory text. It can be longer than one line, may include CHR$(10) (linefeeds) or a backslash, which will be replaced by CHR$(10). Please make sure that the ROM-code contains only CHR$(10) and no backslashes!

length: Length of the string to be entered. If the value 65536 is added to the length, the cursor is placed at the beginning of the string, otherwise at the end. Adding more values to it, gives you further options:

+2^16	position cursor at the start of the string.
+2^18	will force the user to enter a value, an empty string return is not allowed.
+2^20	will accept integers only.
+2^21	will accept fractions and integers.
+2^22	a negative sign is allowed.
+2^23	enables password entry (only asterisks are shown, and the cursor cannot be moved).
+2^30	disables the function of the [ESC] key in this menu.

x_pos,y_pos: These define the position of the window. If these parameters are omitted, the current pointer position is used.

Example:

```
name$=READ_STRING$('READ IN NAME',ram1_suggestion,'Enter
filename or press ENTER',45)
```

VIEW_FILE filename , [start_string] , [end_string] , [width]

This procedure enables the user to view files. This can mean just looking at the file or to show a HELP-type text.

The parameters are defined as follows:

filename: Name of file to be viewed. The entire name must be given.
start_string: If given, the file is searched for this string and displays it from the position where the string is found. Attention: if the string is not found, nothing is displayed!
end_string: If given, text is displayed only until this string is found. The end string itself will not be displayed.
width: The width of the windows in characters (minimum 16). If no width is given, the window is extended to its maximum width.
x_pos,y_pos: These define the position of the window. If these parameters are omitted, the current pointer position is used.

Example:
```
VIEW_FILE flp1_BOOT
```

BUTTON_WAIT name , [x_pos , y_pos]

This procedure applies a button to the current job and waits for the user to use it as a "wake up". The job may not have any open windows, otherwise a correct positioning of the button cannot be guaranteed. If QPAC II's Button Frame is present, the procedure will attempt to place the button there.

The parameters are defined as follows:

name: Name which will appear inside the button or address of sprite definition (size maximum 40x20 pixels).
x_pos,y_pos: These define the position of the window. If these parameters are omitted, the current pointer position is used.

Examples:
```
BUTTON_WAIT Test
BUTTON_WAIT "Press ENTER"
```

BUTTON_SELECT (name , [x_pos , y_pos] , [-1] , [second_sprite], [controlword])

This function applies a button to the current job and waits for the user to use it as a "wake up". The job may not have any open windows, otherwise a correct positioning of the button cannot be guaranteed. If QPAC II's Button Frame is present, the procedure will attempt to place the button there. The return parameter specifies the keystroke which woke the button:

0	ESC
1	WAKE (if second sprite was defined as separate item)
2	HIT (if button immoveable)
3	DO

The parameters are defined as follows:

name: Name which will appear inside the button or address of sprite definition (size maximum 40x20 pixels).

x_pos,y_pos: These define the position of the window. If these parameters are omitted, the current pointer position is used.

second_sprite: There may be an optional second sprite which is placed right to the name or first sprite in the button. Again, the address of the sprite definition has to be given.
The value
+1 prints the WAKE-Symbol,
+2 prints the "Changed - please save" sprite.

controlword: To activate various features, add the values:

+1	Second sprite is separate item - keystroke is WAKE.
+256	Second sprite should flash.
+65536	HIT is valid selection keystroke (button immoveable).
+131072	ESC is valid selection keystroke.

Example:

```
PRINT BUTTON_SELECT (Test,,,3,2,256+131072)
```

GET_DEFAULT$ (key)

allows the user to read all of the pre-defined default extensions and all of the pre-defined default directories, used in the Extension-Select window and the Directory-Select window respectively, as well as some other default settings.

The parameters are defined as follows:

key: inquiry key.

At present, values between 0 and 7 (inclusive) return the default extensions 1 to 8.
Values between 32 and 39 return the default directories 1 to 8.
64 to 66 return the colour combination for the main window, sub-window and buttons, respectively. (Ignored in current version).
96 is the default update-icon flash frequency, used by XBTN, for example.
97 is the default delay for “hint” windows, used by QD (application program), for example.

There is one special key: -1, which returns the version number of the currently installed Menu Extension.

All other values return "bad parameter".

SET_DEFAULT key , value$

allows the user to set all of the pre-defined default extensions and all of the pre-defined default directories, used in the Extension-Select window and the Directory-Select window respectively, as well as some other default settings

The parameters are defined as follows:

key: inquiry key. All the values of **GET_DEFAULT$** are allowed here too.
value$ the value to be set into the default key location

Example:

```
SET_DEFAULT$ 4,”_XXX”
```

LIST_SELECT ([menu_name] , item$, status% , flags , [lines] , [columns] , [x_pos , y_pos])

Allows the selection of a single item out of a list of items. The return parameter is either the index number of the item, -1 for ESC or -2 for OK.
If no status% is supplied, the function returns after the selection of one item. If status% is supplied, the function returns on do or OK only. status% will be updated on return. The first dimension of the item$-array and the dimension of status% have to be the same, otherwise status% is ignored.

The parameters are defined as follows:

menu_name: Title for the window.
item$: A two-dimensional string array containing all items (starting at 0!).
status%: A one-dimensional integer-array, containing the status for every corresponding item. The values are:
- 0 available
- 128 selected
- 16 unavailable
- 16384 can be edited
- 256 (on return) has been edited

flags: for menu-control: a 1 underscores the first character of every items, this is used as selection keystroke, a 4 puts a keystroke in front of the items, and a 16 centres the item. 64 turns a [Hit] into a [Do]
lines: maximum number of lines.
columns: maximum number of columns.
x_pos,y_pos: These define the position of the window. If these parameters are omitted, the current pointer position is used.

Examples:

```
10 DIM vers$(2,12)
20 vers$(0)='Letter'
30 vers$(1)='Small Packet'
40 vers$(2)='Printed matter'
50 PRINT LIST_SELECT('Please select',vers$,,,,3,2)

10 nr=10:DIM name$(nr,20),name%(nr)
20 RESTORE:name%(3)=128:name%(5)=16
30 FOR x=0 to nr:READ name$(x)
40 PRINT LIST_SELECT(,name$,name%,16)
50 DATA 'Paul','Fred','Christopher','Ann'
60 DATA 'Mary','Michael','David'
70 DATA 'Thomas','Christine','Stephanie','Nicholas'
```

SYS_SELECT$ ([menu_name] , [list-type] , flags , [lines] , [columns] , [x_pos , y_pos])

Allows the selection of a single item out of a list of items. The contents of the list depend on the list type parameter:

Hotkey stuffer buffer history (list type 0) lists all the strings, filenames etc. which have been placed into the stuffer buffer history). Please note that as long as there is no or just a current stuffer buffer entry available, no list will appear. As soon as two or more entries exist, the list will show all of them. There is also the possibility of getting lists of all currently running jobs (list type 16), all things available in the whole system (list type 32), and a list of executable things only (list type 34).

The return parameter is the selected item, e.g. stuffer buffer string, job-name, thing-name.

The parameters are defined as follows:

menu_name:	Title for the window.
list-type:	As described above.
flags:	for menu-control: a 1 underscores the first character of every items, this is used as selection keystroke, a 4 puts a keystroke in front of the items, and a 16 centres the item.
lines:	maximum number of lines.
columns:	maximum number of columns.
x_pos,y_pos:	These define the position of the window. If these parameters are omitted, the current pointer position is used.

CHAR_SELECT$ ([menu_name] , [available_bits%] , , [font] , [x_pos , y_pos])

Character select allows the user to select a charcter out of a list of characters or the whole character set.
It needs an optional menu name, an optional availability byte, optional font for the character table sub-window, optional position and optional colour attributes.
If the user escapes via ESC, the return value is a string with zero length. If the user leaves the menu with OK, the return string is set to the selected character.

The parameters are defined as follows:

menu_name: Title for the window.

available_bits%: Define the character range to be used. If 0, the whole character set is used:

+1	non-printable characters
+2	digits
+4	lower-case characters
+8	upper-case characters
+16	remaining printable characters
+64	cursor control characters

font: Address of a standard font, which is used for the character table.

x_pos,y_pos: These define the position of the window. If these parameters are omitted, the current pointer position is used.

The User Side

As the Menu Extension is usually supplied with programs which use the Pointer Environment, the following instructions assume that you are familiar with the Pointer Environment itself, otherwise you should read the section of the documentation first which explains how to use the Pointer Environment (it really should be somewhere in the main manual). Just for a quick reminder: [Hit] means, point to an item and press [SPACE] or the left mouse button, [Do] means, press [ENTER] or the right button while you are over an item.

The FILE SELECT window

The FILE SELECT window is always shown when the user is required to enter or select a filename. Here you can enter the filename either directly or edit a suggested one by selecting the menu option directly beneath the request. Beneath this are two menu options with which you can recall the contents of the HOTKEY buffer and all previous contents. Just select the menu option and the contents will be written to the "suggested" area. Confirm with OK and the input will be accepted by the system. You can also edit the name, of course. Select it with [F3] or point to it and press [SPACE] or [Hit] with the mouse button. When the cursor flashes, you can edit the string. [ESC] brings you back into "pointer mode".

Most of the rest of the window concerns the current drive. You will see two sub-windows, the right hand (larger) one will only show the filenames, the smaller one on the left only the subdirectories. In these windows all the files are sorted alphabetically (unless you configured the sort off). The files will be taken from the current drive and must all have the correct extension, or ending (if any). The extensions for sub-directories are ignored, because subdirectories don't really have extensions. Now you can edit the drive and/or the ending. Depending on how you configured the Menu Extension, you will either get the full (complete) file- and directory-name, or just anything which comes behind the current directory or device name (e.g. if the current directory is WIN1_TEXT_ and it contains the files WIN1_TEXT_FRED and WIN1_TEXT_PAUL, only FRED and PAUL will be shown).

You can get quickly into the file-list subwindow by pressing [TAB], or into the subdirectory-window by pressing [SHIFT] [TAB].

If you press [F2] or [Do] at the directory menu option, a further window is overlaid, from which you can select pre-defined devices and sub-directories (see Directory-Select below). If you just select a directory, the list of files will be updated. You can also "update" it by selecting the lightning symbol or by WAKEing up the job to which the window belongs, should this be hidden.

If you press [UNDO] or [CTRL] [ALT] [Cursor UP] then the "Ext:" menu option will be cleared, so that all files will be shown. If you [Do] the "Ext:" menu option, then a window overlay will show some suggested extensions (see EXTENSION SELECT below). If you press [F4] you will get directly into the extension select menu, regardless of the current extension setting. If you press CTRL 0, CTRL 1 or CTRL 2(does not work on any keyboard layout, sorry!) then you will see TYPE 0, TYPE 1 or TYPE 2 instead of an extension. Only those files are listed which match the filetype: TYPE 0 is used for most files including text files, TYPE 1 shows executable programs only, TYPE 2 all relocatable files (which usually end in _REL).
If the extension starts with an underscore, then this underscore is automatically internally converted into a "." for DOS disks (e.g. the extension is _TXT, but if you look at a DOS-Disk, it will list all files ending with .TXT).

Above the current directory is a list of available devices, e.g. WIN or FLP. There are also drive numbers from 1 to 8 listed. To select FLP1_, press [F] and [1], and the directory window displays FLP1_. If a filename is already suggested in the F3-Filename-location, e.g. ram1_test_bas, then a [Do] on one of the devices will automatically change the device in the filename field as well, e.g. in the example above, a [Do] on WIN will change the filename to win1_test_bas.

Above the file list you will see an arrow "<-". By selecting this option you can retrace a step back along the subdirectory tree without having to edit anything. So if you're in directory FLP1_TEXT_PAUL and select this once, then you get to FLP1_TEXT_. The next time you select it, you get FLP1_. If you select a subdirectory in the lefthand subwindow, then you'll "get in", i.e. the name will be taken over for the directory and the file and subdirectory list read in again.

But to get back to the list of files: You can select any file you like. [SPACE] or [Hit] accepts the name as "suggested" and enables the cursor, so that you can edit it. [ENTER] or [Do] takes the name and carries out an OK automatically. If the window is too small to show all the suitable files, the normal scroll arrow will appear in order to scroll the next batch of names up the screen. You can also select files or directories by pressing the character which is in front of the name.

At the right you will see a scrolling bar. Move the mouse pointer to this area, [Hit] and the area will be shown relative to the total area. [Do] at this area and the window will split, enabling you to control the two parts indepently from each other. Move to the split and [Do] to join the window together again. If you are already in the filelist-subwindow and you press [TAB], the cursor jumps over the bar. Another [TAB] brings you back to the centre of the filelist. For the directory-subwindow, [SHIFT] [TAB] does the same.

The file-select has got two other facilities: you can get a list of the whole directory tree, i.e. all sub-directories from the current level downwards are "expanded" into all filex they contain. This gives you a better overview. But beware, the tree option may take a long time, especially on higher levels on heavily filled harddisks. Therefore, tree is de-selected automatically when you "move up" in the tree.

You can also have a quick look at the file before you select it for load. Select the View item first, then hit the file you want to view. You can view it now until you press [ESC]. If you think this is the right file, select OK 'cause the filename has moved to the suggestion. You can also [Do] while you are over the filename, as [Do] ignores the state of the view item.

You can construct a new filename very easy: choose the required device and subdirectory in the left part of the window. Then, make sure the filename extension is right. If you press [F5] now, the current directory and the current extension will be used as the "suggestion" and the cursor is placed exactly between directory and extension. All you have to do now is: type in the missing part of your filename. If you would like to save a file called "win1_source_qd_fred_asm" and you are already in the directory "win1_source_qd_" and the extension is "_asm", then just press [F5] and type in "fred".

Every filename selected with [Do] or [OK] will also be stuffed into the HOTKEY Buffer, so that it can be immediately selected from somewhere else, or the next time you pop up a file-select window.

And this brings us to the HOTKEY Buffer area of the window: There you'll find two items: Current and Previous. The HOTKEY Buffer stores selected filenames (e.g. if you select files in QPAC2 or save a file in QD etc.) so that you can access the filename later with a single click. If you [Hit] the Current item, the current hotkey buffer contents are brought into the filename window. If you [Do] the Current item, the current contents are used directly, the file with the filename stored therein is loaded.
The HOTKEY buffer also stores a number of previously selected filenames. If you [Hit] the Previous item, then it cycles through the previously stored names every time you hit it. If you [DO] the Previous item however, then you get a list of all the previously stored filenames (provided there is a list). Otherwise, nothing happens.

Finally, there is the [F10] Fileinfo II item. Fileinfo II is freeware, and you can obtain it from many sources. You should read the Fileinfo II manual which comes with Fileinfo II and configure it to suit your needs. What [F10] does is: it will call Fileinfo II with the currently selected filename (in the F3 – Filename item) so that Fileinfo II can carry out any action configured for it.

The DIRECTORY SELECT window

This window allows the selection of different drives resp. subdirectories.

The user can pre-set the eigth most used subdirectories; they can be selected in the left sub-window. Use the MENUCONFIG program to pre-set the subdirectories on the MENU_rext file. If one of the preset directories is hit, it moves to the suggestion at the top of the window. If it is DOne, this selection is returned to the calling program. If the selected subdirectory does not exist, the device is searched for matching names in higher levels. If, for example, you select FLP1_BACKUP_TXT_ and there is no such directory on FLP1_, then only FLP1_ is shown. If the directory BACKUP exists, you will see FLP1_BACKUP_.

The currently installed drive systems (MDV, RAM, FLP or WIN, for example) are selectable immediately, together with the eight different possible drive numbers. If selected, the suggestion changes and the list of subdirectories is read in again.

The right sub-window displays the subdirectories which exist in the current directory level. This window behaves exactly in the same way as the subdirectory-window in FILE-SELECT does.

[F2] allows you to edit the current suggestion.

"DATA Default" sets the suggestion to the current setting of the system's data default directory, which is usually set using the DATA_USE command, or QPAC 2's SYSDEF, for example. The same applies to "PROG Default".

To configure the subdirectories please use the program MENUCONFIG and configure the file MENU_rext.

If the Menu Extension itself is not built into ROM or EPROM, but loaded into RAM, you will see an "Edit" item being available. If you select this item you can edit all of the eight pre-defined directories and save a new configured version of the Menu Extension directly afterwards. Press ESC here if you do not want to save it, or give the filename. If you modify the suggested filename for the Menu Extension, this modification is also saved together with the new preset directories.

The EXTENSION SELECT window

This window serves to select the most usual file extension. Move to a menu option and press [SPACE] or [ENTER] to select. You can also use the number keys [1] to [8] to select the menu options. All ten extensions may be set by the user using MENUCONFIG, or, if the Menu Extension is in RAM, by selecting the Edit item - it behaves exactly like the Edit facility in DIRECTORY SELECT (see above). 'No' returns no extension, i.e. an empty string.

The ITEM SELECT window

When you see a window with one to three menu option, you can make your selection by pressing [SPACE] or [ENTER]. You could also press the first letter of the option.

The READ STRING window

You are asked to enter a string or filename. Under certain circumstances you may be offered a suggested name. You can either press [ENTER] to accept the suggestion, edit it as usual using the cursor keys or just enter a new string.

The VIEW FILE window

This window enables you to view a file. You can scroll one line by pressing [SPACE] or the left mouse button while the pointer is over the view-window. You can scroll one page by pressing [ENTER] or the right mouse button. The lightning-symbol lets you start again viewing the file from the beginning. Selecting WRAP causes any line, which exceeds the permitted width, to be continued on the next line.

The BUTTON

If a program is in Button mode, then you can wake this up by moving to the button area and pressing [ENTER]. If the button is not positioned inside the button frame, you can move the button by using [SPACE]. In some buttons you will also see a lightning symbol next to the name. This symbol allows you to WAKE the program. If you see a flashing disk-symbol right to the name, then this is used to remind you that you have changed the data with this program, but not saved this modified data. Some buttons also react on [ESC], which removes the button and the corresponding program.

Configuring the Menu Extension

We suggest that you use MenuConfig to configure the settings in the Menu Extension.

You will find three configuration blocks: **General**, **Directory Select** and **Extension Select**.

General contains the following configurable items:

- **Filename of Menu Extension:** fairly self-explanatory. If you modify the Menu Extension (using the Edit item in some menus), you can save the modifications to this pre-defined filename.
- **Flash-Frequency for update icon:** used by XBTN, for example. You can pre-define the behaviour of the flashing icon in four steps, from steady (not flashing) through three flashing speeds. The setting can be read and overwritten with the INFO and SET calls in machine code, or GET_DEFAULT$ and SET_DEFAULT in BASIC).
- **Delay for hints:** here you can specify a timeout for little hint windows to pop up (if the program supports hints, like QD does). Previously called “icon explain”, but “hint” is the better term.
- **Alphabetic order in File-Select:** very self-explanatory. We cannot even remember why this was made configurable, as an alphabetic order seems to be the most useful thing here.
- **Selection keystrokes in File-Select:** dates back to old times, where the screen was limited to 512x256 and users wanted to get more information into the window.
- **Short filenames in File-Select:** also dates back a long time. For level 2 devices it must be set to “no” – otherwise it can result in crashes!

Directory Select: contains the eight pre-defined directories for the Directory select menu. The setting can be read and overwritten with the INFO and SET calls in machine code, or GET_DEFAULT$ and SET_DEFAULT in BASIC).

Extension Select: contains the eight pre-defined file extensions for the Extension select menu. The setting can be read and overwritten with the INFO and SET calls in machine code, or GET_DEFAULT$ and SET_DEFAULT in BASIC).

The Scrap Extension

The Scrap Extension provide the routines to support cutting and pasting between application programs using the Scrap. It can also be used to transfer data within an application. All data that is cut or copied gets into the Scrap. If it is text, then it may overwrite the previous contents of the Scrap, or it may be added. Graphic always overwrites the previous contents. In case of text data the Scrap is a kind of a large Stuffer Buffer.

The Scrap is designed to transfer relatively small amount of data. Text size is limited to +32767 characters (standard QDOS string), but may contain <LF>'s to separate lines.

The Scrap Extension comes with the resident extension MENU_rext. It adds the thing 'Scrap Extensions'. The first try to put data into the Scrap creates a new thing, called 'Scrap'. The contents of the Scrap are kept until the Scrap is removed or new data overwrites it.

Using the Scrap Extension is done in the same way Menus is used. The Extensions currently implemented are:

CLR Clear Scrap and return memory to system.
INFO Returns information about the state of the Scrap and its contents.
PUT Overwrites or adds data.
GET Reads data from Scrap.

The routines to use and free the Scrap Extensions need the same parameters which Menus need. You will find the routines for doing this in the library, the names are UT_USSCP and UT_FRSCP.

The fourth character of Extension IDs which have only three characters must be SPACE.

There are five types currently defined:

text	0	standard SMSQ/E string
sprite	2	standard sprite definition
blob	4	standard blob definition
pattern	6	standard pattern definiton
area	8	standard partial save area

CLEAR SCRAP (CLR)

This routine is used to return the memory used by the Scrap to the System. No parameters.

SCRAP INFO (INFO)

Returns information on current scrap. If the Scrap Thing does not exist, the type returned is -1. In this case, the other two parameters are returned undefined.

The format of the parameter list:

```
sci_dtyp    equ     $04     current type
            thp.ret+thp.ubyt
sci_mcnt    equ     $0c     scrap modify count
            thp.ret+thp.uwrd
sci_bufl    equ     $14     buffer length minimum for current scrap contents
            thp.ret+thp.ulng
```

SCRAP PUT (PUT)

This is the SCRAP PUT function. It is used to add or overwrite the contents of the Scrap. If the Scrap does not exist, then it is created. If the type is 0 (string), then scp_flln should be zero to overwrite the previous contents or non-zero to add it to a string which is already in the Scrap. If the action to put the code into the Scrap is needs more code than just copying a string, a routine may be supplied to do this job.

The format of the parameter list:

```
scp_dtyp    equ     $03     type of data to write
            thp.opt+thp.ubyt
scp_strg    equ     $08     source string (if type=0) or base address of
                            graphic data
            thp.call+thp.str+thp.ulng
scp_flln    equ     $0c     add string flag (if type=0) or length of data
            thp.opt+thp.ulng
scp_uccd    equ     $10     user-supplied copying code
            thp.opt+thp.ulng
```

SCRAP GET (GET)

This is the SCRAP GET function. It is used to read the contents of the Scrap. If the code to copy the data has to be complex, the address of a user-supplied copy-code may be given. In this case, scg_dest and scg_bufl may be 0.

The format of the parameter list:

```
scg_dtyp    equ     $03     type of required data
            thp.opt+thp.ubyt
scg_dest    equ     $04     destination address for data
            thp.opt+thp.ulng
scg_bufl    equ     $08     length of destination buffer
            thp.opt+thp.ulng
scg_uccd    equ     $0c     user-supplied copy-code
            thp.opt+thp.ulng
```

User-supplied Copy-Code

Entry		Exit
D0		0, +1 or error
A5	ptr to Scrap Thing	preserved

All registers except D0 must be preserved

The structure to which A5 points is:

```
scp_type  equ     $08     b   type of current SCRAP contents
scp_mcnt  equ     $0a     w   scrap modify count
scp_maxs          equ     $0c     l   maximum size of scrap
scp_ussz  equ     $10     l   current size used
scp_data  equ     $14         data
```

SuperBASIC Interface

There are new commands which allow to clear the Scrap and add contents. Data may be read using new functions.

SCRAP_CLEAR

removes the Scrap from the Thing list and returns the memory used by it to the system.

SCRAP_PUT [0],string$,[addflag]

puts text into the Scrap. The first 0 is the type, which may be ommitted (0 is default).

The parameters are defined as follows:

string$: The string to be inserted or added.
addflag: If ommited or 0, the string$ overwrites the previous contents. If 1, string$ is added to the previous string.

Examples:

```
SCRAP_PUT 0,'This will be put into the Scrap'
SCRAP_PUT ,' and this will be added.',1
```

SCRAP_PUT type,address,length

puts any kind of graphic objects into the Scrap.

The parameters are defined as follows:

type: The types currently allowed are:

sprite	2	standard sprite definition
blob	4	standard blob definition
pattern	6	standard pattern definiton
area	8	standard partial save area

address: Memory start address of grahic object.
length: Length in bytes of object.

SCRAP_LEN

is a function which returns the size of memory which is necessary to keep the current scrap contents.

SCRAP_TYPE
returns the type of the current scrap contents.

SCRAP_CNT
returns the scrap modify count, to check whether the scrap contents have changed or not.

SCRAP_GET$
is a function which returns a string which contains the contents of the scrap, if it is type text. If not, an empty string is returned.

SCRAP_GET
is a function which returns an address with a copy of the current scrap contents, if it is graphical data (type <> 0). If not, the returned parameter is 0. It is the user's responsibility to release the memory (e.g. SuperToolkit's `RECHP addr` or `CLCHP`).