EasyPtr 4

Manual

Revision 1.03

CONTENTS

(Parts 1 & 2 Base version and SuperBASIC Toolkit)

1.0 Foreword to the free release

It's been 10 years since the release of EASYPTR 4 and this is apparently enough time to forget the pain I had creating it. So when I was asked how to obtain EasyPtr 4 I thought it's maybe time for releasing it to the public. If you find it useful you are still allowed to send a donation my way, of course ;-) But if you ever create anything useful with it I'd like at least a copy of the software. The manual is pretty much as-is with only minimal changes like removed commercial informations. The rest of the distribution has not changed at all.

Marcel Kilgus, March 2016

2.0 Foreword to Release 4

It's probably been 10 years since the last EASYPTR release. Quite a long time, but as the PE basically also hasn't changed most of that time people didn't miss out on much. This changed with the advent of the the colour drivers and my work on the new WMAN 2, that includes many new features and possibilities. The PE finally got much ahead of EASYPTR, so an update was due but the original author, Albin Hessler, didn't have the time to do it anymore.

At one time I came in possession of the sources and just for fun and to see whether I can do it extended them a bit to have better colour support, but this was by far not mature enough for release in the wild. Many months and much bugging by some people later I'm proud to present a real update to EASYPTR. In the end I've put much more work than I've ever envisioned into it and I truly hope that you will enjoy it. Most updates concentrated on the "heart" of EASYPTR, namely EASYMENU. It now possesses fully functional colour support and also some features that Albin wanted to do for the original EASYPTR but never got around to doing.

On other parts like the BASIC toolkit I have tried to fix all issues people have found over the years and extended them when needed.

EASYMENU

- Full colour support
- Much improved colour choosing facilities
- Full support of the new "system palette" facility that gives PE applications a common look and which provides very easy "colour theme" support for all menus
- Full support for big objects (sprites), which have previously been limited to 32kb
- Fully resizable and movable main menu
- Full handling of scaling flags to very comfortably allow the creation of resizable menus
- Some fixes of old problems people had with EASYMENU
- ...and much more

EASYSPRITE

- Fully compliant with the GD2 graphics drivers (it will still only design QL colour sprites! For the creation of high colour sprites have a look at bmp2sprt, sprted or pngconv, they are all free software with the last one being from yours truly, alas it's Windows software).
- In high colour mode it can be used to design both mode 4 and mode 8 sprites (this was just an interesting thing to do, I truly hope nobody is still developing for mode 8!)

APPMAN

- Many changes to make it compatible with big objects (up to 16MB!)
- Uses a graphically much improved interface when running in a high colour mode (courtesy of Wolfgang Uhlig)

Basic extensions

- Fixed all problems that have been reported to me and that I was able to reproduce
- Fixed problems regarding the new colours and the handling of resizable menus
- New keyword WSIZE to allow implementing standard menu resizing behaviour

EASYSOURCE

The assembler source of this application is basically too difficult to maintain. Some changes have been made for it to be able to handle new definitions, but ultimately I hope some 3rd party will eventually come up with an alternative.

- Instead of crashing it will now output extended sprites as one big "dc.w" block

Marcel Kilgus, April 2005

3.0 Introduction

EASYPTR is a development package for Tony Tebby's Pointer Environment. This operating system extension to QDOS enables mouse and pointer driven menu and program control.
To avoid misunderstandings: EASY means that the structures of the Pointer Environment are easily built up with EASYPTR. For SuperBASIC it means that a menu driven program can be written with the extensions in this package with the minimum of effort.
EASY does not mean that only simple applications can be generated.

EASYPTR is not a general programming tool, and especially no course of introduction to programming for beginners. To use the structures generated by EASYSPRITE and EASYMENU you must have at least basic programming knowledge in SuperBASIC or Assembler. Without this you might not be able to write the program that is intended to be controlled by the Pointer Environments features.

The first English version of EASYPTR Version appeared in summer 1990. Since then the package has been further developed and extended.
Version 2 basically has completely revised SuperBASIC extension with about 55 commands, and an enhanced menu generator EASYMENU with many new tools and features. For simple applications and beginners now a base version is available.
Version 3 provides additional features to support the new Library Routines in Part III and has all known bugs fixed.
Version 4 was done by a new author, Marcel Kilgus, and tries to bring the tools in sync with all the latest developments in the SMSQ/E world, especially all colour enhancements.

Base Version:

The menu generator, EASYMENU, allows the development of menus in a true WYSIWYG style on-screen.
The sprite generator EASYSPRITE enables sprites, blobs and patterns to be designed or generated (cut out) from screen pictures. The SuperBASIC extensions EASYBMEN contain all basic Commands to set up and control a menu.
SuperBASIC Toolkit
APPMAN allows the management of the appendices to SuperBASIC extension files in a very comfortable way.
The SuperBASIC extensions EASYPTR. EASYMEN PTRMEN have more than 50 commands to control pointers and menus.
Definitions generated from EASYSPRITE and EASYMENU can be appended to the extension files. Thus, together with Qliberator, compact jobs can be generated, which are as fast as assembler written programs in setting up and controlling menu structures and pointers

Library Routines

These routines are intended to be called from and linked together with modules generated from a C compiler or assembler. The routines work together with definitions generated from EASYMENU and EASYSPRITE. For special purposes EASYSOURCE can be used to disassemble any complete menu or sprite definitions (note: support for high colour sprites is limited, they will only be included in a menu definition as one big block).

My intention when writing the EASYPTR package was to give access to the excellent features of the Pointer Environment (which make QDOS an up to date operating system of same or higher standard, compared with other modern or modernised operating systems for other machines) to as many programmers as possible. Only if programs in an up to date standard are written, QDOS and its descendants will have a chance to survive.

Albin Hessler, Aichtal in February 1993
Re-edited and reformatted, Q Branch, Portslade 2002.
Updated and re-edited, Marcel Kilgus, Denkendorf 2005

3.1 Copyright

Kingston upon Thames
KT2 6PJ
England

The copyright of all other parts of the EASYPTR package is owned by
Albin Hessler Software
Im Zeilfeld 25
D - 72631 Aichtal
Germany

All enhancements for EASYPTR release 4 are owned by
Marcel Kilgus
Katharinenstraße 25
D - 73728 Esslingen
Germany

3.2 Guarantee

This release is made public as-is. No guarantee is given whatsoever.

3.3 Disks

EASYPTR is available as a single ZIP file containing everything that was previously shipped on two separate 3 1/2" DD disks:

boot
bootqpac2
easymenu_exe
easysprite_exe
easybmen_cde
easyext_cde
easysource_exe
appman_obj
easyptr_cde
easyptrr_cde
easymen_cde
easymenr_cde
ptrmen_cde
ptrmenr_cde
putget_cde
plus several demo files and utilities.

3.4 System requirements

The EasyPTR 4 development tools are only guaranteed to work on SMSQ/E v3.10 or later. The resulting applications should still work with older systems and other operating systems, provided that the environment supports all features used in the application.

For standard QLs there is the serial mouse driver SERMouse available from us. This driver works with most standard PC serial two or three button mice. You can also use the QIMI Mouse or the superHermes system from T.F. Services.

3.5 System configuration

The simplest way to use EASYPTR is to make a working copy of disk 1 (see above), copy the Pointer Environment files (ptr_gen, wman and Hot_rext) onto that disk and then restart the computer with the boot program.

If your system is already built on the extended Pointer Environment (e.g. if you have SMSQ/E installed, which we strongly recommend), you only need to install the latest versions and the EASYEXT extensions to be able to use the programs .
The executable programs, EASYMENU, EASYSPRITE may then be used.
The file 'bootqpac2' is an example how to install QPAC2 together with the EASYPTR System.

3.6 Program configuration

Most of the EASYPTR files are configurable. Use either the Config or Menuconfig program and follow the prompts.

4.0 Basic topics

If you are not yet familiar with the pointer environment features, you should read chapter 11 first. The basic structures that the pointer interface and the window manager are able to deal with are described below.

4.1 Graphics objects

4.1.1 Blobs

A blob definition provides a mask through which a pattern is visible. Thus a blob must always be assigned a pattern to combine with. The blob definition consists of size, origin and the bit mask that defines any pixel through which the pattern is visible.
A blob might be the outline of a letter. If it is combined with a white pattern, it will be a white letter, if it is combined with a red/green striped pattern, it will be a red/green striped letter.
Blobs can be generated with EASYSPRITE.

4.1.2 Patterns

A pattern defines the colour of any pixel. The pattern may repeat in x- and y direction. The pattern definition consists of x- and y- repeat distance, where the x-repeat must always be a multiple of 16 (the number of pixels in a 32-bit long word), and the colour pattern bit mask, which defines the colour of each pixel. Pattern definitions may be generated with EASYSPRITE, where the x-repeat distance is automatically set to the next multiple of 16.

4.1.3 Sprites

A sprite is a small picture that may be drawn stationary or moved around with the mouse or the cursor keys. The sprite definition consists of size, origin (which is used to position the sprite when it is drawn stationary, or which gives toe point pointer position if the sprite is used as a pointer sprite), pattern mask (blob) and pixel wise colour pattern.

Sprite definitions can be linked together to form a dynamic sprite. If such a definition is set as a pointer sprite, the interface will show any of the linked definitions for a certain time, and then the next. The effect is a dynamic object.
The time for which any shape is visible may be set individually. The maximum period is about 5 seconds.
Sprites can be generated with EASYSPRITE

4.2 Menu elements

A menu is a list of options to control a program. While a command line controlled program must be given the correct command by typing it in, the menu offers the program options on the screen to select them. The selection is done by moving a pointer sprite over the corresponding object and selecting it with the selection key (HIT or SPACE). Selecting with the DO or ENTER key will normally start the action.

The pointer environment offers standardised structures to be managed by the window manager.

The menu generator EASYMENU is used to set up this structure on the screen and generates the corresponding binary definition automatically. The structure elements, which can be used, are as follows.

4.2.1 Main window

The main window describes the area of the screen, within which the menu structure is drawn. The main window is defined by size, initial pointer position, background colour, border size and colour, shadow, pointer sprite and pointers to lists of information sub-windows, loose menu items and application sub-windows.

4.2.2 Information sub-windows

Information sub-windows may be used for general information, colour design or partitioning of the menu. They may be situated in any place within the main window also overlapping any other elements. They are defined by size, origin, colour, border size and colour and a list of information objects as required. Info-windows may also be used for general purposes of the program, e.g. to write out temporary information, but info-windows are not concerned by the menu request of the window manager.

4.2.2.1 Information objects

Any information window may have several info-objects. These are arranged in a list. Info-objects are defined by the origin position (relative to the info-window), their size and the object itself, which may be a text string, a stationary drawn sprite definition or a blob/pattern, pattern/blob combination. As for text, a letter may be assigned to be underlined and also the character size and ink colour are defined. Graphical objects are drawn with their origin situated at the given origin position within the window.

4.2.3 Loose Menu Items

Loose menu items can be situated at any place within the main window: Normally they do not overlap each other or any application sub-window. They are defined by their size and origin. The paper colour of all loose menu items is defined as an attribute of the main window for three different states, available, selected and unavailable. Loose menu items are identified by their object, which may be a text, a sprite, a blob or a pattern. For the latter ones, the respective combination partner is also set as an attribute for all loose items for the three states. Any loose item also has a selection key. For text objects a letter can be assigned to be underlined (normally the selection key).

Loose menu items are directly connected with a program routine, called the action routine. This routine is automatically called by the interface, when the item is selected (either HIT or DO). The routine either does the work itself or just controls the item status and initiates a return to the main program with appropriate control data.

The latter principle is used by the EASYMEN SuperBASIC extensions, as SuperBASIC procedures and functions cannot be called from within machine code routines directly. Here the SuperBASIC Program must decide from the item number, which program procedure shall be called.

4.2.4 Application sub-windows

Application sub-windows may be situated at any position within the main window. Normally they should not overlap other application sub-windows and loose menu items, as then the menu request will not work properly. That's why EASYMENU does not allow overlapping.
Application sub-windows are defined by their size, origin, colour, border size and colour, selection key and pointer sprite. If no special pointer sprite is set, then the pointer of the main window is used.
Application sub-windows may be used for general application purposes of the program, e.g. for text editing, drawing ... Application sub-windows are part of the menu request, i.e. if the pointer is within an application sub-window, as for loose items an action routine (the hit routine) is called which may take over program control.
Application sub-windows may also have a regular menu, which can be larger than the window. Hidden parts are scrolled or panned into the visible window area.

The menu items are arranged in a rectangular grid, where the x size is the same for all items in a column, but the column width can be different for any column, and the y-size is the same for all items in a row, but the row height can be different for any row. Thus the menu shape is defined by the column/row sizes and the distance (spacing) between columns/rows. While loose menu items are always selectable, application sub-window menu items are only selectable if they are visible and if the pointer is within the window. All items have their own action routines, which is called when the item is selected.

4.3 Scalable menus

WMAN has two means of supporting the resizing of menus: multiple menu layouts in a single menu definition and the scaling of a specific menu layout. When a menu of a certain size is requested, WMAN first searches for a fitting layout and then, if the definition is scalable, scales it to exactly fit the requested size.

While multiple layouts are not supported by EASYMENU (though you can of course design several different menus and emulate this functionality this way), starting with EASYMENU 4 freely scaleable menus can finally be created. This is accomplished by using so called "scale flags". A scale flag is part of every size and every origin value within a menu definition and can have a value from 0 to 4.
Previously all elements were hard coded to a scale of "0", which simply means "don't scale at all". In the new EASYMENU all values can finally be specified.
Basically the value of the flag determines how much of the absolute size change of a window (relative to the size the window has been designed in) will be added to the size/origin itself:

Flag value	**Proportional size change**
0	none (default)
1	¼
2	½
3	¾
4	full (1:1)

Example: a loose item has an X origin with a scale flag of 1 and an X size with a flag of 3. When the window width gets increased by 100 pixels, then 25 will be added to the origin and 75 to the size of the loose item.

Always remember to also set the scaling flag of your main layout (usually 1:1 for both size components), otherwise the menu will not increase in size at all!

The scale flags for the different elements can be entered in the list mode menu.

Some example code for a resizable menu:

```
100 DIM pr%(16)
110 ch = FOPEN("con_")
120 MDRAW#ch; "example"
130 :
140 REPeat main
150  k = MCALL(#ch; k, 0)
160  SELect ON k
170  = -2: REMark Resize window
180     PVAL#ch; pr%
190     WSIZE #ch, x%, y%
200     xs% = pr%(8) - x%
210     ys% = pr%(9) - y%
220     MCLEAR#ch
230     MDRAW#ch; "example", -1, -1, xs%, ys%
240  = -3: EXIT main
250  END SELect
260 END REPeat main
270 :
280 MCLEAR#ch: CLOSE: CLAMP: CLEAR: STOP
```


5.0 The sprite generator

EASYSPRITE **file 'easysprite_exe'**

With EASYSPRITE sprites, blobs and patterns up to a size of 64 x 48 Pixel in MODE 4 or 32 x 48 Pixel in MODE 8 can be designed on the screen (these are the maximum sizes allowed for pointer sprites by the pointer interface not older than V1.36). The definitions are automatically saved in the binary form required by the pointer interface. EASYSPRITE is mainly intended to deal with small graphical objects and for this offers some simple tools. EASYSPRITE is not a design program! Graphical objects can also be designed within a design program. EASYSPRITE has an option to import screen cuts and convert them to the pointer interface readable binary form of a sprite definition.
Several sprite definitions may be linked together to form a dynamic sprite.
EASYSPRITE has a working area, which always offers the maximum size, while sprites are stored and saved in their maximum extent only, independent of the position within the working area.

5.1 Main menu

The main menu is divided in preview window, working area and menu items for option selection and program control.

5.1.1 Preview window [V]

In the top left corner is the preview window. Here the sprite is immediately drawn stationary in its real size any time a pixel is set. If the pointer is moved into the preview window, the sprite can be tested as a pointer. The background colour of the window is initially set to 0 = black, it is increased by. 1 on a HIT and by 8 on a DO in the window. TAB will reset the colour to black. Thus the sprite can be tested over every background colour. This is especially useful if the sprite has transparent arts.

5.1.2 Working area [P]

Any rectangle of the grid in the working area represents a pixel. A pixel is set to the preset colour by a HIT on the corresponding rectangle of the grid in the working area. DO works as a rubber and will reset to transparent. A pixel may not be set or reset twice, but only if another pixel has been set between. This allows horizontal and vertical lines-to be drawn with a little experience. Also areas can be filled with the same colour when the HIT button or SPACE is kept pressed and the mouse / cursor keys moved.

It's not easy to draw a perfect circle in the working area, as the x/y proportion is not 1:1. Here it is advised to draw a circle in a Basic window with the CIRCLE command and to import it with the 'Cut out' option available in the 'Files' menu. It may be convenient to have some often used ground forms saved in a library.

The sprite (frame) in the working area has three speeds. Type TAB to change the speed.

5.1.3 Menu items

[Quit] **[Esc]**

Remove the job. Confirm on request.

[ZZZ] **[CF1]**

If the QPAC2 button frame is present, the job will be made a button.

[Erase] **[E]**

The sprite in the working area is erased (all pixels set to transparent). The designed sprite is lost if it was not saved to a file or stored in the internal store area. Confirm on request.

[Mode] **[M]**

Change screen mode. Objects that shall be used in the same shape in both mode 4 and mode 8, must be designed in mode 8, using only the four main colours.

[Colour selection] **[0] - [8]**

Upper row: Select the actual pixel colour.

These items are selectable with the number keys [0] to [8], thus the colour may be changed by pressing the colour number (8 for transparent) while the pointer sprite is over the working area.

[SHIFT 0] - [SHIFT 8]

Lower row: Select re-colour colours.

To any pixel colour a re-colour colour may be assigned. If a re-colour item is selected with DO or ENTER, all pixels are re-coloured as selected.

[Re-colour Reset] **[R]**

The reset item (at the right of the colours) will reset the re-colour items to the natural order.

[-] **[-]**

Set the whole current definition to transparent (remove ground) (see → [Ground][+]).

[Ground] **[G]**

A pixel of a sprite definition may be set to have a not transparent (white) background, i.e. as if it where drawn on a white paper. It will always appear in the same colour.
Or it may be set to have a transparent background, then the pixel colour will be XORed with the background colour, when the object is drawn.

Selected: The current pixel is set to have a (white background.
Available: The current pixel is set to have no (back)ground (i.e. transparent)

[+] **[+]**

Set the whole current definition to not transparent. (add ground)
(see → [Ground] [-]).

[Files] **[F]**

Call the [FILES) menu for load and save operations d the [Cut out] option.

[Store] **[S]**

Call the [STORE] menu.

[Origin] **[O]**

Set the sprite origin. Move the pointer in the working area to the desired position and HIT.
Normally the origin of a sprite definition should be positioned within the sprite area.
The origin of a sprite is referred to, when the object is drawn (e.g. see → BLOBW or SPRW) and to determine the pointer position, i.e. the pointer position returned by RDPT or PVAL is always the position of the sprite origin.

Important hint: If you get an 'out of range' error message in EASYMENU or after BLOBW or SPRW, then a bad origin position might be the reason. The area within which an object shall be drawn, must be as large as the area covered by the object and it's origin, which can be outside the set pixels.

Exception: Objects in menu items are always drawn relative to their extent. The origin position is ignored here.

[Fill] **[I]**

Turn on fill option. Move the pointer over a closed area (an area which is surrounded by not transparent pixels of different colour) and HIT. All pixels will be reset to the actual colour.

[90°] **[9]**

The sprite in the working area will be turned by 90° clockwise around the origin. The transformation is an un-scaled, simple pixel-to-pixel mapping. All pixels that fall outside the working area will be lost.

[↑_] **[\ ]**

The object in the working area is moved to the top left corner.

[→] [←] [↓] [↑] **[→] [←] [↓] [↑]**

The object in the working area is displaced by one pixel in the selected direction.

5.2 Menu FILES

File names can be edited in the line at the bottom of the menu. If the menu extensions (menu_rext) are present and EASYSPRITE has been configured to use them, then the FILE SELECT menu is presented. The default directory may be omitted, also the ending. If the file cannot be opened on the given directory, then the Toolkit 2 DATA_USE default is tried.

5.2.1 Menu [Save as]

This menu is presented after a save option has been selected. Here is determined whether the object shall be saved as a sprite, blob or pattern definition.

Sprite: The object is saved as a sprite definition as it appears.
Blob: Only the pattern mask of all non-transparent pixels of the object is saved.
Pattern: Only the colour pattern of the definition is saved. The x-repeat distance is automatically set to the next multiple of 16. Therefore you should be aware that the x-size of the object is divisible by 16 to get proper patterns. The y-repeat distance is set to the y-size of the object.

Attention:
Blob and Pattern definitions cannot be loaded in EASYSPRITE as they only have a part of the information that is needed. Therefore you should always save the same object as a sprite definition too.

5.2.2 View window

Here a sprite is drawn statically when its number is set in the [Number >] window. If the pointer is moved in the view window, then the sprite is shown as a pointer sprite. The background colour is initially set to 0 = black. On a HIT in the window the colour in increased by 1, on a DO it is increased by 8 .

5.2.3 Menu items

[ESC Quit] **[Esc]**

Return to main menu.

Mode [4] [8] [8+4] **[8] [4]**

The actual mode is shown. In Mode 8 the option [8+4] is available when the actual definition in the working area shall be saved (sprite number 000). When [8+4] is selected two linked definitions for both modes are saved. This definition is about 25% shorter as if the both definitions are saved separately. The pointer interface always looks after a definition for the actual mode in a chain of linked definitions. Stored sprites (numbers from 001 upwards) are always saved with the whole chain of definitions linked with them.

5.2.4 Save

[Number] **[1]**

The object that is set here is saved. [000] is the object in the working area. The object is automatically restricted to the smallest size that contains all set pixels.
The other numbers refer to stored objects (see → menu [STORE]).
Set the number by a HIT (one step) or DO (step 10) on the scroll arrows (up or down). The number may be edited directly, when the number window is hit.

If several definitions are linked together, then the whole chain is saved, beginning with the actual number.
Definitions can be linked together to have shapes for different modes and/or to form a dynamic sprite.
If a definition was stored with the [8+4] option, then it is saved in the same way.
The internal sprites [i1] to [i7] may not be saved directly, but must be fetched via the [Fetch] option in the [STORE] menu before.

[Single] **[S]**

Saves the sprite definition or chain of definitions to a separate file with ending' _spr', ' _blb' or '_pat'. The type and ending are selected in the [Save as] menu.

[Collection] **[C]**

Saves the definitions) to a special collection file with ending '_msc'. If the file is found, the definition is appended to this file otherwise a new collection file is created. In this file, objects are saved in a special format. Multiple Sprite Collection files ('_msc') are intended for use in the menu generator EASYMENU (e.g. to have all sprites, blobs and patterns for a specific menu in one file) and may be loaded there.

Sprites saved to a collection file cannot be reloaded to EASYSPRITE and should therefore be saved as single definitions.

Part II only:
Collection files can also be created and managed in APPMAN.

[Append] **[A]**

A sprite, blob or pattern definition (or chain of definitions) is appended to a file.
It can only be appended to SuperBASIC extension files of type EASYBMEN (base version), EASYPTR, EASYMEN, PTRMEN or EASYAPP (part II only).
The files may have other names as they are identified internally. Appending to other files is not possible.
Appended sprites are identified by their name (max. 8 characters, no underscores) that is set with the next menu option.
Appended sprites cannot be reloaded to EASYSPRITE and should therefore be saved as single definition too.

Part II only:
Appendices can be managed with APPMAN.

[Name] **[N]**

The sprite name can be edited when this item is selected. The name may have up to 8 characters (no underscores!) and is always converted to lower case automatically.

[Load] **[L]**

A single sprite definition ('_spr') is loaded. The definition is stored into the internal store area and linked into the list of stored sprites after the actually set number (see → [Number]). Loaded sprites are fetched into the working area via the [Fetch] option in the [STORE] menu.

[Cut out] **[U]**

Call the Load screen dump menu.
Please note that this menu is only available if EASYSPRITE is run in QL colour mode.

5.3 Menu [Load screen dump]

With this option sprites can be cut out from any screen dump or the windows of an actually running job.
Three different screen dump formats can be loaded:

Standard QL screen dumps 512x256 pixels
Screen images in the pointer environment format
Atari black and white screen dumps 640x400 pixel
or
a jobs window

This feature is combined with a scan option through all actually loaded jobs windows. The scan option also works when a screen was loaded, but then this screen is lost.
QL screens must have the ending '_scr', the others the ending '_pic'.
A QL screen dump must be exactly 32768 bytes long, an Atari screen dump exactly 32000 bytes. The size of a pointer interface format screen cut is in the header information of these files.
The screen is loaded and the cut out operation is started.

Cut out option control:

Keys [+] and [-]

The primary windows of all actually running jobs are scanned up or down. This option also works if a screen has been loaded before, but then the screen is lost.

Frame

A pointer in the size of 64x48 pixels appears. It can be moved with the mouse or the cursor keys. If HIT or SPACE is held down and the pointer is moved, the size will change.
Put the frame in the desired size over the part of the screen that shall be cut out. Then proceed as follows:

DO or **ENTER** saves the part of the screen that is within the frame as a sprite. The frame disappears. Just move a little to see the effect.

a second **DO** or **ENTER** will save the sprite in the [Store] area.
a **HIT** or **SPACE** will, instead, give up the sprite (it is not stored).

After both options the frame will appear again.
You can now make the next cutout or quit by pressing **ESC**.

ALT + CURSOR ←→↑↓

If a screen is larger than 512x256 pixels, you may scroll/pan in steps of 256/128 pixels.

ESC

Quit the cut-out option

5.4 Menu STORE

Up to 247 definitions can be stored in the internal store area. (depending on the available memory size)

5.4.1 View window

Here a sprite is drawn statically when its number is set in the [Number >] window. If the pointer is moved into the view window, then the sprite is shown as a pointer sprite. The background colour is initially set to 0 = black. On a **HIT** in the window the colour is increased by 1, on a **DO** it is increased by 8.

5.4.2 Menu items

[ESC Quit] **[ESC]**

Return to main menu.

Mode > [4] [8] [8+4] **[8] [4]**

The actual mode is shown. In Mode 8 the option [8+4] is available. When [8+4] is selected two linked definitions for both modes are stored. This definition is about 1/4 shorter as if the both definitions are stored separately. The pointer interface always looks for a definition for the actual mode in a chain of linked definitions. If sprites are to have the same shape both in mode 4 and 8, they must be designed in mode 8 but only the four colours available in Mode 4 (black, white, green, red) must be used.

[Store] **[S]**

Saves the sprite designed in the working area to the internal save area. Up to 247 definitions can be stored.
The sprite is linked into the list of stored sprites after the number shown in the number window (this may be of interest when dynamic sprites are designed).

Number window **[1]**

The number is selected by a **HIT** (one step up or down) or a **DO** (step by 10) on the scroll arrows (up or down). Only numbers up to the highest stored number are allowed. The number may also be edited directly from the keyboard, when the number window is selected.
The number determines the definition

- which is shown in the view window [View]
- which shall be fetched into the working area [Fetch]
- which shall be removed [Remove]
- behind which the definition in the working area shall be stored [Store]

[View] **[V]**

With [View] selected, a stored sprite definition, **whose number is set** in the number window, will be drawn in the view window, and will be set as the pointer sprite of that window. When the view window is **HIT** the background colour (number) will be increased by one. A **DO** will increase by 8.
The 8 internal sprites supplied with the pointer interface are shown downwards from 0 (marked [i1] - [i7]). Beyond these a scan option [scn] begins to run. On any **HIT** on the down arrows, the memory is scanned for the next sprite definition.

[Fetch] **[F]**

The selected stored sprite is loaded into the working area. Even the internal sprites and scanned sprites can be fetched.

[Remove] **[R]**

The selected stored sprite is deleted and unlinked from the list of stored sprites. When the number is 0, all stored sprites are removed. The internal sprites cannot be removed.

[Link] **[L]**

Remark: The two number windows below [Link] are managed in the same way as described above. Their selection keys are [P] and [N].

Two sprite definitions are linked together. In the header of the first sprite a pointer is set to point to the other sprite, thus the pointer interface may follow this chain. When two sprites in mode 4 and mode 8 are linked, the pointer interface will select the correct version automatically. To link two sprites in different modes, [Time] must not be selected! Several sprite definitions may be linked to form a dynamic sprite, when additionally to the link a time is set. If such a chain of linked definition is set as a pointer sprite, the interface will show each definition until its time has elapsed. The time is stepped by 20 ms. Thus the period of a dynamic sprite may be up to about 5 seconds (255*20ms).

[Time] **[T]**

If [Time] is selected with **HIT** or **SPACE**, then on the next [Link] the time, as set in the time unit window (key [M]), will be set for the definition in the left number window below [Link] (time for link from definition).
If [Time] is selected with **DO** or **ENTER**, the time, as set in the time unit window, is set for the definition in the right number window below [Link] (time for link to definition). This must always be done for the last definition of a chain of
definitions or if the time for a single definition in a chain must be altered.

See following example.

Example to generate a dynamic sprite definition:

You have designed seven sprite definitions and stored each of them with the [Store] option. These will be linked together to form a dynamic sprite with a period of about 1 second. Then any definition must be visible:

1000 ms / (7 * 20 ms) = ca.'7 time units

Set the time in the time unit window to 7.
Select [Time] with **HIT** or 'T' .
Then:
1st step:

Set the left link number to 001,
set the right link number to 002

and select [Link].

The two definitions are now linked together and the time is set for the left one.

[Link] is now unavailable until a new number is set.

2nd step:

Set the left link number to 002,
set the right link number to 003

and select [Link].

The two definitions are now linked together and the time is set for the left one.

[Link] is now unavailable until a new number is set.

and soon...

6th step:

Set the left link number to 006,
set the right link number to 007

and select [Link].

The two definitions are now linked together and the time is set for the left one.

[Link] is now unavailable until a new number is set.

5.4.2.1 Finally

select [Time] with **DO** to set the time for the last definition (number [007]).

You've got it?

No? Then just try it. It would really be a pity if you could not use this marvellous option. It's really worth to spend some time with it. There is a definition 'hase_spr' on disk 1 as an example for a dynamic sprite.

Yes? Then you should read the following

Remark:

In the above example the time was set to the same figure for all definitions. The time may be set individually for every single definition! The chain must not follow the internal store numbers, i.e. you may link 2 to 7, 7 to 3, 3 to 9 etc., though you might get confused!?

EASYSPRITE will not allow circle links.
When a definition of a chain is removed, the chain is interrupted. So it is possible to remove a single definition, change it's shape, and re-link it at the same position. Sprites can be stored at a pre-selected position (see → [Store]). The new definition must then be linked with the previous definition and the following definition.

6.0 The Menu Generator

EASYMENU **file 'easymenu_exe'**

With the menu generator menu definitions can be designed on the screen. The generated menu definition is automatically saved in the binary format readable by the window manager. Menu definitions can be reloaded, i.e. they can be changed and adjusted during program development interactively. If you are not yet familiar with the different menu elements, please read chapter 4.

6.1 The Working Area

The working area covers the whole screen. The working area is overlaid by the main menu. When one of the items [Set up], [Remove] or [Change] is selected, the main menu will disappear and the menu window in the working area is shown on the selected background. The initial background colour can be configured with 'config'. It can be changed with the options [Background] [Neutral] and [Basic].
The menu window may cover the whole screen.

6.1.1 The Tools of the Working Area

Cancel **[ESC]**

To return to the main menu at any time. The actually selected option is cancelled.

Target sprite

In options [Remove] and [Change] a dynamic target sprite is shown. This sprite must be moved over the item which shall be removed or changed. A HIT will select the item. Only items of the pre-selected type can be chosen

Active Sprite **[TAB] [0] [1] [2] [3]**

When the item which shall be processed is determined, a sprite will blink in the bottom right corner (the active corner) of this item. The item is linked to this sprite and may be moved around with the mouse or the cursor keys. When HIT or SPACE is held down at the same time the size is changed. With DO or ENTER the actual size is confirmed and the operation is closed.

The change rate may be set to four speeds (see → config) available through keys [0] to [3] or switched around with the TAB key. The active sprite blinks according to the change rate (slow see → slow move or change, ...).

Speed 1:	2 pixel in x-direction 1 pixel in y-direction
Speed 2:	4 pixel in x-direction 2 pixel in y-direction
Speed 3:	8 pixel in x-direction 4 pixel in y-direction
Speed 0:	is the same as speed '1' with an additional timeout. This timeout may be decreased (see → faster) with the '+' key or increased (see → slower) with the'-' key (15 steps, preset to 8). The chosen timeout will be kept as long as the job is running and is not configurable (as it depends on the actual hard- and software constellation).

By this an easy pixel wise positioning is possible even with the keyboard or the fast QL-Atari.

Fine Justification **[F] [G]**

In option [Outline] [Change] the window limits may be moved relative to the contents with the fine justification.
Type [F] and then move the window relative to its contents while **HIT** or **SPACE** is pressed.
[G] returns to the normal move/change option.

Confirmation **[DO] or [ENTER]**

The actual layout is confirmed and the program will return to the main menu.

Warning signal

Menu items and application window may not overlap. If such a constellation is confirmed, a warning signal is send out and the-confirmation is not accepted.

Error report 'out of range'

If the size of menu items is changed when an object is already set, it may occur, that the object does no longer fit into the element. In such cases an 'out of range' error will be reported. The menu will not be drawn completely. The error can be resolved, by making the size of the item large enough again, or by removing the item.

6.2 Main Menu

The main menu is presented directly when the job is started. It is removed for any operation on the working area.

6.2.1 Menu items

[ESC Quit] **[Esc]**

The program is terminated. A safety box will request you to confirm.

[Sleep] **[CF1]**

If the QPAC2 button frame is present, the program will be made a button.

[Size] **[CF3]**

This button is in the lower right corner and can be used to resize the EasyMenu main window. Hitting this item with DO will result in the window being maximized, if it's not already occupying the whole screen, otherwise it will revert to its previous size.

[Move] **[CF4]**

If the main window does not cover the whole screen then it can be moved around using this button.

[Erase] **[E]**

The working area is cleared, the whole menu structure, all data and loaded sprites will be lost definitely. Confirm on request.

[Mode] **[D]**

When EasyMenu is run in QL colour mode this button can be used to switch between QL mode 4 and QL mode 8.
In high colour mode it has a different meaning, there it switches between the 4 different system palettes (0-3) and one additional, built in palette (4). With this you can quickly see how your designed menu looks with different colour schemes. The current palette number is shown in the mode button.

[Files] **[F]**

Call [FILES] menu.

[Outline] **[U]**
[Info window] **[I]**
[Info object] **[T]**
[Loose item] **[L]**
[Appl. Window] **[W]**

Select the type of the menu element to be processed. Only one of these types can be active.

[Set up] **[S]**

Unavailable in [Outline] option!
A new element of the pre-selected type will be generated.
Depending on configuration, on **HIT/DO** or **DO/HIT**

- control is passed to the working area directly
- or the menu [{element} LIST] is called

In the working area an element of the same size as the last element generated, or of the minimum size 2x2 pixel, is created in the middle of the main window.
In the list, an element can be generated by setting values in the four items for size and origin in row 0. When all four values are entered, [OK] becomes available. The new element is generated when [OK] is selected.

[Remove] **[R]**

Unavailable if type of element is [Outline] !
An element of the pre-selected type can be removed. As above ([Set up]) control is passed to the working area or the list.
In the working area the element is selected with the target sprite. It will then be removed immediately.
In the list, the element can be selected in the first column, then a HIT on the [Remove] item in the LIST menu will remove it.

[Change] **[C]**

An element of the pre-selected type can be changed. As above ([Set up]) control is passed to the working area or the list.
In the working area the element is selected with the target sprite. Then it can be treated as before.
In the list, the element can be selected in the first column. A HIT on [Select] will pass control to the working area, where the element is directly presented.
This works also for hidden elements.
In the list changes can also be made by typing new values directly in the items for size and origin. These values are immediately set! Only possible values are accepted.

Attention:
When changes are made by typing new values in the list, there is no test on overlapping elements. By this you can bypass the restriction that overlapping loose items and application sub-windows are not accepted in the working area.

[Attributes] **[A]**

Call ATTRIBUTES menu.

[Object] **[O]**

Call [OBJECT] menu.

[Applw. Menu] **[M]**

Call [Application sub-window menu] menu.

[View] **[V]**

The main menu disappears and the working area is shown. Type any key to return to the main menu.

[Pointer position] **[P]**

Set the initial internal pointer position in the menu window. The interface will try not to move the pointer, when the menu is drawn. The pointer sprite which is set for the main window (outline) will be written at this position.
Move the pointer sprite to the desired position and confirm. If the pointer is outside the outline of the main window, the pointer position will be set to the top left corner. If the pointer sprite does not fit into the window size, the standard pointer sprite (arrow) will be set automatically. (see → menu [FILES] [Position][Relative][Fixed])

Background

[Neutral] **[N]**

Selected with a HIT:
The whole background of the working area is set to the default (see → config) or the previous set colour. Selected with a DO: A pattern with all available SMSQ/E palette colours is shown on screen. The background colour may be selected.

[Basic] **[B]**

The actual SuperBASIC window contents is set as the background of the working area. Thus the menu may be positioned relative to the existing SuperBASIC windows (e.g. a pop down menu may be positioned within a previously designed main menu).

6.3 Menu {element} LIST

If [Set up][Remove][Change][Attributes][Object] or [Applw. menu] is selected, the element list can appear. It depends on configuration whether the list appears on **HIT** or on **DO**. Selecting [Direct] will immediately switch over to the working area, where the menu elements can be set up, changed, removed or selected with the mouse pointer. It is possible to create a menu with the list alone.

6.3.1 Menu items

[ESC] **[ESC]**

Return to main menu.

[OK] **[DO] or [ENTER]**

Only available in [Set up] after all 4 values (x/y-size, x/y-origin) have been set.
Create a new element.

[Direct] **[D]**

Go to working area.

[Remove] **[R]**

Only available if the list was called from [Remove] in the main menu.
Remove element selected in column [No.].

[Select] **[S]**

Go to working area. The element selected in column [No.] is directly presented.

Column [No.]

To select a certain element with **HIT** or **SPACE**.
If selected with **DO** or **ENTER**, a new position can be entered
The list is then sorted with the element in the given position.
The numbers in the list are the element numbers as used by the SuperBASIC extensions (e.g.. MCALL function value). Negative for loose items.

[X] [Y]

The big entry windows show the X/Y object size, which can directly be changed here. The smaller windows to the left of the respective values are for the window scaling functionality (new in EASYPTR 4). The values entered can be between 0 and 4, meaning that the particular value will change by n/4th of the overall size change of the window, i.e. if a window width is increased by 100 pixels and the X size has a scale flag of 2, the item width will be increased accordingly by 2/4th (= 1/2) of that (= 50 pixels). See chapter 4.3 for more information.

[X0] [Y0]

The origin of the object. The same scaling flags are available as in the [X][Y] case. The scaling flags for the origin of the main layout are actually used on the pointer position, as a menu definition does not have a window origin!

[Object]

Shows the type of the object (in case of a graphics object like a sprite) or the first few characters for text objects. By selecting it the object can be changed.

6.4 Menu LIST OF ALL OBJECTS

When the items [Object] in the [OBJECT] menu, [Blob →][←Pattern] in the [ATTRIBUTES] menu or the column [Object] in the element LIST menu are selected, the menu with the list of all actually available objects will appear. The window tries always to be large enough for the biggest object in the list. An object already set for the specific element will appear selected, else the first object in the list will be presented.
The type (text, sprite, blob, pattern) of the object is indicated. If OK is **HIT**, the actually visible object will be set (but not in {element) list menu).
Per default the menu is already filled with all the system sprites available (i.e. move, resize, sleep sprites, various mouse pointers and more). The appereance of these sprites will vary from system to system as the user can exchange them.
Graphic objects (sprites, blobs, pattern) must have been loaded with the [FILES] menu load option before.
If a menu definition has been loaded, all objects in this menu are in the list.
If a text object has been set with the [Text input] option in the [OBJECT] menu it is also available in the list.
If the same text shall be used several times in the same menu, it should be reused via the list.
All objects can be multiply used. They are only once in the definition. This is specially useful with sprite definitions, as it may save a lot of memory.
Remark: Objects are automatically removed from the list, if the last item where they are set is removed (but not if they where loaded with the [Collection] option).

6.4.1 Select Object

The list of objects can be scrolled up and down with the scroll arrows and bars. It's possible that an objects is not visible, then selecting it may help. To set an object, select [OK]. [ESC] will return to the calling menu, without change.

6.5 Menu FILES

File names can either be edited in the edit line at the bottom of the main menu, or are selected via a File-Select menu if the menu-extensions menu_rext are loaded and EASYMENU was configured to use them.
The default directory (see → configuration) and the ending can be omitted. If the file can not be found on the given directory or the-default directory, then also the Toolkit 2 DATA_USE directory is checked.

6.5.1 Menu items

[ESC Quit] **[Esc]**

Return to main menu.

Mode [4] [8] **[4][8]**

Will force a mode for the menu (see → MDRAW). If neither [4] or [8] is selected, the menu will be drawn in the actual window mode.
If a menu shall be used in both mode 4 and mode 8, it must always be designed in mode 8.

[Save] **[S]**

The menu definition in the working area is saved to a file. The ending ' men' is added automatically.

Part II only:

Menu definitions which shall be disassembled with EASYSOURCE must always be saved as a single file.

[Append] [A]

SuperBASIC only

Menu definitions can be appended to the extension files of type EASYBMEN (base version), EASYMEN or PTRMEN (part II only).
Thus menu definitions used by a certain program must not be loaded from a separate file, and a compact job may be generated when compiled with Qliberator. The extension file must then be linked into the compiled job with the

REMark $$asmb=... option.

The correct instruction parameters can be found at the beginning of the extension files. Actually it is always:

REMark $$asmb=filename,4,82

The appended menu is identified by its name. The name is set by selecting the [Name] item.
As appended menu definitions can not be reloaded it is advised to save the menu definition separately to a ' _men' file (see above).
The extension files must not have the original names, as they are identified internally.

Attention: Never append to the original files.
Part II only: Extension files with appended definitions can be managed with APPMAN. Here it is also possible to recover single definitions from an appendix.

[Relative] [R]

SuperBASIC only

Normally the Pointer Interface will try to avoid changes of the current pointer position. A menu will be drawn such that the current pointer position must not be changed, i.e. the window origin will be positioned relative to the internal pointer position and the actual absolute pointer position is kept unchanged.
Example:
The internal pointer position is set to position 20, The actual absolute pointer position is 100,50. Then the menu is drawn with its origin at position 80,40. The actual absolute pointer position is unchanged. The pointer will appear at the internal pointer position 20,10. As usual, there are exertions from this rule. If parts of the window would fall outside the screen, the pointer position is adjusted accordingly.

[Fixed] [F]

SuperBASIC only

The menu will be drawn:

- at the actual absolute screen position which it has in the EASYMENU working area if the menu channel is a primary window.
- at the actual absolute position relative to the origin of its primary window, if the menu channel is a secondary window.

The primaries outline must be large enough that the menu fits into the window.
When a position is selected with the [Basic] option, the origin of the SuperBASIC #0 outline must be moved to the top left corner of the screen (part II only: see → WMOV).

[Name] [N]

Appended menus are identified by this name. The name can have up to 8 characters (no underscores) and is converted to lower case automatically.

Load **[M][C][P][B][T]**

Load a file:

type	**ending**
menu	_men
(in the format generated with EASYMENU)	
collection	_msc
sprite	_spr
blob	_bib
pattern	_pat
(all in the format generated with EASYSPRITE)	

A menu definition is loaded into the working area.
Attention: An existing menu in the working area will be removed and is lost.

Objects are added to the list of objects. They are then available in the [OBJECT] or [ATTRIBUTES] menu.

When loading a menu, instead of a filename you can also specify the memory address of a window definition (by entering the hexadecimal address prefixed with a $ sign, i.e. $123456) or you can try to snatch the definition of the window of a running job by entering the hexadecimal SMSQ/E channel ID prefixed by a # sign, i.e. #01230008 for channel number $8 with the channel tag $123. This has no correlation to the SuperBASIC channel numbers! Also, it's not alyways possible with every job as some do not have complete window definitions but construct at least parts of the window at runtime.

With version 4 all memory space limits have been removed, thus EasyMenu should even be able to handle huge menu definitions or sprites occupying anything from several kilobytes to some megabytes. Please note however that AppMan and appendixes in general are limited to 16MB per object. Considering that appendixes are always held in memory such big objects are probably not a good idea anyway, though.

6.6 Menu ATTRIBUTES

All menu elements have certain attributes. These are set with this menu. Only the specific attributes of an element are available.

6.6.1 Menu items

[OK] **[DO] or [ENTER]**

Confirm values. Return to main menu

[Window]	**[W]**
[Menu item available]	**[A]**
[Menu item selected]	**[S]**
[Menu item unavailable]	**[U]**

These items are only available when [Outline] or [Appl. window] is selected. Here is selected for what the colour selection shall be made.

[Border width] **[H]**

Only possible border sizes are accepted.

[Shadow depth] **[D]**

Only available for the main window outline. Only possible values are accepted. In contrary to the pointer interface in general (see → OUTL), the window manager accepts only regular shadows at the right and the bottom side. A value of 1 is equivalent to a 4 pixel x-shadow and a 3 pixel y-shadow.

[Character size] **[Z]**

Only available for Info-objects. All values possible for CSIZE are allowed. If an objects partly falls outside its info-window after a change of the character size, the window manager draw routine will break with an error. The menu is then drawn incompletely.
Remedial measures: Reset character size to old value.
Remove info object
Change info window size

Colours

[Paper] **[P]**
[Ink] **[I]**
[Border] **[B]**

Set the corresponding colour:

Colour values:

- may be edited from the keyboard directly, if the appropriate item is selected.
- all values higher than 255 (i.e. the new high colour modes) are always shown in hexadecimal notation, prepended by a $ sign.
- If the edit window is left with a cursor up or down (mouse up or down + **HIT**) the colour value can be selected from several different colour palettes. The colour palette can be switched using the right hand menu items:
  - **Sys**: Shows a scrollable list of all currently defined system palette entries. Please note that all new menu elements already use the appropriate system palette colours by default, i.e. an information window will automatically have "InfWin bg“ as paper colour. Only using **Sys** colours ensures that you can easily support several colour schemes and even make your window automatically show up in the user's prefered colours.
  - **Aur**: This is a palette with the 256 fixed colours that the Aurora graphics board (and thus mode 16) use.
  - **Fix**: This, by default, looks the same as **Pal**, the SMSQ/E palette, but results in fixed RGB colours instead of a palette index. RGB colours are guaranteed to look the same on all systems (hardware permitting), whereas the SMSQ/E palette can be redefined.
  - **Pal**: The counterpart to **Fix**. The colours here depend on the system EasyMenu (and ultimatively the designed menu) are running on. Per default it looks the same as the **Fix** palette, but **Pal** can be redefined by the user.
  - **Gry**: All gray shades from black to white.
  - **QL**: all the traditional QL colours in the range of 0 to 255.
  - **Bdr**: only selectable when chosing a border colour. Shows a collection of the available 3d border styles. As some new border types do not match the height and width of traditional QL borders, and thus may lead to undesired effects, there are several compatibility modes available. A red part in the graphics means that while the border will occupy this additional space, it will not be filled but left untouched. A blue part means that the space will be filled with the underlying *paper* colour. Green lines are just to show a window circumference for reference.

[Blob →] **[L]**
[←Pattern] **[T]**

Set the combine partner. Blob for pattern objects, pattern for blob object.
Any blob needs a pattern to be combined with and vice versa.

Menu Items

If a blob is set as a menu item object (e.g. a full circle) then for the three possible item states, a pattern must be set to signal the different status (e.g. a white pattern for available, a red pattern for selected and a red/white stipple for unavailable). These pattern are set as an attribute for all menu items.

If a pattern is set as an object, the three blobs to combine with MUST be set as an attribute (e.g. a black pattern and a square blob for available, a square blob with a cross in it for selected, ...).

Info Objects

A blob or pattern must have been set via the [OBJECT] menu before ties partner to be set becomes available here.
Attention: As the interface would stop with a 'bad parameter' error if there is no combine partner for a blob or a pattern to be found, EASYMENU, by default, sets a 2x1 pixel blob and a white pattern

6.7 Menu OBJECT

Set the object (text, sprite blob or pattern) in info window objects or menu items.
Set window pointer sprite for application sub-windows or main window.
Application sub-windows will use the pointer of the main window if no extra pointer sprite is set. By default always the standard small arrow is used, which must not be set.

Text objects can be edited with the [Text input] option. All other types of objects must be loaded (see → FILES).
All objects can be used several times.

6.7.1 Menu Items

Only those items which are relevant for a specific type of element are available.

[Quit] **[ESC]**

Return to main menu.

[Text input] **[I]**

Only available for info objects and menu items.
Control is passed to the working area, where a window, widened by one character, over the element is presented for text input.

Justification

[Center]	**[E]**
[Left]	**[L]**
[Right]	**[R]**
[Top]	**[T]**
[Bottom]	**[B]**
[Number windows]	**[H] [V]**

Objects in menu items can be justified. The justification is preset to [Center], i.e. a text or sprite will be written in the center of the item. To justify top, bottom, left or right a positive value must be set. The value must be edited in the number windows. This is the distance in pixels from the item border. All objects are justified relative to their extent, i.e. the origin position in sprite or blob objects is ignored.

[Object] **[O]**

Call menu LIST OF ALL OBJECTS (if there are some available).

Key **[K] [C]**

To determine the selection key of a menu item or an application sub-window.
Select [K] and press the key directly at the keyboard. Letters will be transformed to upper case.
Select [C] for an input of the ASCII-code (0 - 255).
Some codes are transformed by the pointer interface to special event keys:

action	key	ASCII code	event key
do	ENTER	10	2
cancel	ESC	27	3
help	F1	232	4
move	CF4	245	5
change size	CF3	241	6
sleep	CF1	233	7
wake	CF2	237	8

SuperBASIC only:
The menu draw routine MDRAW will automatically set action routines for the following events, if the event key is set as selection key:

code (selection key)	routine
3	remove menu (like MCLEAR)
4	window move
7	sleep (If QPAC2 is present)

Codes 27, 245 and 233 are converted to 3, 5 and 7 by MDRAW, but no action routine is set. Thus the items will accept ESC, CF4 and CF1 but return from MCALL is normal, thus the program can do its own routines.

[Underline] **[U]**

If set to a positive value n, the n'th character will be underlined. Normally the selection key is taken.
If the value is 0, no character will be underlined.

6.8 Menu Application Sub-Window Menu

Here the menu structure of an application sub-window can be set up. !t is possible to select
No menu
Full menu
and **Only bars.**

In option [No menu] the colours for the arrows and bars can be set. This is useful if the window shall be used for a dynamic menu structure.
In option [Full menu] the complete static menu structure can be set up. This option can also be used to set up a dummy menu, only used to get the right window proportions for a dynamic menu in runtime.
The option [Only bars] is intended for assembler programmers, it is of no use for SuperBASIC.

Remark:

The menu in an application sub-window is organised in a regular column/row grid. The hit size of an item and the spacing between items can be set equal for all items in a column/row or can be set to individual values for any column/row. There are several lists for which, depending on size, memory is allocated dynamically. Thus the size of a menu is only restricted by the available memory. On the other hand, if the lists have been set up, the size can not be changed. That's why the size can only be set once and then the item becomes unavailable. The size can then only be changed by removing the menu. So be sure, that the size is OK before you set other data (objects etc...)

Remark:

Pan and (or) scroll bars can only be drawn by the interface if the window can be widened on the right side by 8 pixels for a scroll bar and by 5 pixels at the bottom for a pan bar. If possible this is done by the window manager automatically. Bars are only visible if the colours are different from the background, so **always set the colours for the arrows and bars first**. Current versions of the window manager (up to 1.39) do not allow to set the bar background colour. The window paper colour is taken instead.

SuperBASIC only:

For dynamic menus set up at runtime with MAWDRAW the window is made smaller by the width of a pan or scroll bar if necessary. This is to make sure that the bars always can be drawn. So eventually you should make the window accordingly wider (8 pixel) or higher (5 pixel) in EASYMENU

6.8.1 Menu COLOURS

Here the colours for pan and scroll bars and arrows are set. Initially all values are set to the system palette defaults. If you do want a scroll/panable window but no bars with it, set the bar ground and bar section colours to 0 (i.e. black). The colours can also be set if no menu structure is set up, e.g.. if the window will be used for a dynamic menu at runtime.

6.8.1.1 Demo Window

The actual colours can be seen at the demo window.

6.8.1.2 Menu items

[ESC] **[Esc]**

Return to menu [Application sub-window menu], nothing is set.

[OK] **[DO]**

Set values and return.

[-] [←] [+] [→] **[-] [+]**

Decrease or increase colour number.
HIT or **SPACE** change by 1
DO or **ENTER** change by 8.

Please note that the bar section is actually defined as a *border* colour. While all colours in the demo window will show up correctly, certain values (i.e. 3d borders) will not show up right in the big colour preview window.

Pan	**[Arrows]**	**[A]**
	[Bar ground]	**[O]**
	[Bar section]	**[B]**
Scroll	**[Arrows]**	**[R]**
	[Bar ground]	**[D]**
	[Bar section]	**[S]**

Select the type for which the colour selection is to change.

[Columns] **[L]**

Number of columns. The item is set to unavailable after number input. The menu size can only be altered by removing the whole menu structure.

[Sections] **[E]**

Maximum number of x-sections.
The number is:

0 if all columns fit into the window width

and is forced to be:

1 if the menu must be panable, because the window is not wide enough.

If >0, the number can be set to any value between 1 and 10, i.e. this is the maximum number of sections which assigns how often the menu can be split.

[Origin] **[O]**

X-pixel origin of the menu. The origin is forced to be at least 12 if the menu is panable in order to allow pan-arrows to be drawn. But more is possible.

Panable menu **[x]**

[Number] **[M]**
[Width] **[W]**
[Spacing] **[S]**

With this application sub-window menu for the columns, the column hit size (item width) and spacing (distance from the origin of an item to the origin of the next item) are set. These values can be the same for all columns (regular spacing), then the first column **[All]** of the set up menu **is selected** and all other columns are unavailable.
If **[All]** is **deselected**, then all other columns become available and individual values can be set.

The spacing must give room for the item border, which is set in the ATTRIBUTES menu with the [Menu item available] option

Both width and spacing can have scale flags like all other objects. In this case the flag (a value from 0 to 4) is prefixed to the actual value and separated by a colon, i.e. a value of "3:100" means "100 pixels, scaled by 3/4". For more information on scale flags see chapter 4.3.

[Rows] **[R]**

Number of rows. The item is set to unavailable after number input. The menu size can only be altered by removing the whole menu structure.

[Sections] **[T]**

Maximum number of y-sections.
The number is:
0 if all rows fit into the window height
and is forced to be:
1 if the menu must be scrollable, because the window is not high enough.
If >0, the number can be set to any value between 1 and 10, i.e. this is the maximum number of sections which assign how often the menu can be split.

[Origin] **[I]**

Y-pixel origin of the menu. The origin is forced to be at least 6 if the menu is scrollable in order to allow scroll-arrows to be drawn. But more is possible.

Scrollable menu **[Y]**

[Number] **[U]**
[Height] **[H]**
[Spacing] **[A]**

With this application sub-window menu for the rows, the row hit size (item height) and spacing (distance from the origin of an item to the origin of the next item) are set. These values can be the same for all rows (regular spacing), then the first row **[All]** of the set up menu is selected and all other rows are unavailable.

If **[All]** is **deselected**, then all other rows become available and individual values can be set.

The spacing must give room for the item border, which is set in the ATTRIBUTES menu with the [Menu item available] option.

Both height and spacing can have scale flags like all other objects. In this case the flag (a value from 0 to 4) is prefixed to the actual value and separated by a colon, i.e. a value of "3:100" means "100 pixels, scaled by 3/4". For more information on scale flags see chapter 4.3.

7.0 The Appendix Manager

APPMAN File **'appman_obj'**

With APPMAN menus, sprites, blobs and patterns appended to the SuperBASIC extension files 'easyptr_cde', 'easymen_cde' or 'ptrmen_cde' can be managed. More, APPMAN may produce an own file format ('_app), mainly intended to store the appendix separately, but also giving access to the definitions through a SuperBASIC command (APPAn).
APPMAN is intended to copy appendices from one extension file to another (e.g. new version), but also offers the opportunity to delete, sort or recover single definitions from an appendix.

APPMAN's _app files can be included in Qliberated programs using
REMark $$asmb=filename,0,60

ATTENTION

Appman only works if the Menu extension 'menu_rext' is loaded.
Sleep only works if the QPAC2 button frame is present.

7.1 Base file

The first extension file loaded with the option

[Extension (+Appendix)]

in the FILES menu is treated to be the base file, which is written out with the appendix when saved with the same option. On all subsequently loaded extension files with appendix, the extension code is thrown away and only the appendix is merged into the list.
Thus a new version of the extension file must be loaded as base file before any other (note: this has been relaxed a bit in version 4.02, where the code of the latest loaded extension file that does not contain an appendix will be used).

7.2 File types

Appman accepts the following file types

Extension files	**original file name**
EASYMEN	easymen_cde easymenr_cde ptrmen_cde ptrmenr_cde
EASYPTR	easyptr_cde easyptrr_cde
Appendix / collection files	**with ending**
EASYAPP	_app
EASYMSC	_msc

The file names may be changed. The files are identified by other means. To all files of type EASYPTR or EASYMSC only sprites, blobs and patterns may be appended.

Single definitions	**with ending**
	_men
	_spr
	_blb
	_pat

Single definitions are only identified by the ending. An internal identification is not present.

7.3 Main menu

Standard functions

Move window, Sleep (with QPAC2 only), Wake (rebuilt list) and ESC (Quit program).

[Files]

Pop up FILES menu.

[Show]

Switch on show mode. If [Show] is selected, menus and sprites are shown if they are selected in the first column [no.] of the list.

[Clear]

Clear all loaded files.

7.3.1 List

[no.]

All loaded definition are numbered through here. To select a single definition for Save/Append operations. If [Show] is selected the definition is shown. Press a key to return to the menu.

[type + no.]

Type of definition (menu, sprite, blob, pattern). Menus and graphical definitions are numbered separately here. If selected a new number can be edited. The object is then moved to the new position. **If '0' is entered, the definition is removed from the list.**

[length]

Length of the definition in bytes.

[name]

Name of the definition. The name can be edited. Separately loaded sprites, blobs and patterns don't have names. Type a name if they shall be part of an appendix.

7.4 Menu FILES

Standard functions

Move window and ESC (Break, no action)

[OK]

Do selected option.

[Load]

Select load.

[Save]

Select save.

[Append]

Select append.

[Extension (+Appendix)]

Load: An extension file of type EASYPTR or EASYMEN is loaded. All appended definitions are merged into the list. The first file loaded is taken to be the base file. It's name is shown.

Save: If a base file exists, it is saved. If it's type is EASYMEN, then all actually loaded menu, sprite, blob and pattern definitions are appended to this file. If it's type is EASYPTR, then only sprite, blob and pattern definitions are appended.

Append: If a single definition is selected in the list (column [no.]), then this definition is appended to the specified file, else all definitions are appended. The file must be of the appropriate type.

[Appendix [_app]]

Load: A file of type EASYAPP is loaded. All appended definitions are merged into the list.

Save: A new EASYAPP file is created. The APPAn command is part of this file, where 'n' is the number shown in the number window. All definitions are appended to this file

Append: If a single definition is selected in the list (column [no.]), then this definition is appended to the specified file, else all definitions are appended. The file must be of the type EASYAPP.

[Command see → APPA] [n]

Here the number of the APPAn command in an EASYAPP file is set. 'n' can be any number from '0' through '9', i.e. ten different EASYAPP files with different commands APPA0 through APPA9 can be generated.

[Collection [_msc]]

Load: A file of type EASYMSC is loaded. All sprites, blobs and patterns are merged into the list.

Save: A new EASYMSC file is created. All sprites, blobs and patterns are saved in this file. EASYMSC files are intended to contain all definitions that are needed for a specific menu and may be loaded in EASYMENU.

Append: If a sprite, blob or pattern definition is selected (column [no.]), then this definition is appended to the EASYMSC file, else all definitions are appended.

[Menu [_men]]
[Sprite [_spr]]
[Blob [_blb]]
[Pattern [_pat]]

Load: A single definition of the specified type is loaded and merged into the list.

Save: Only available if a single definition is selected in column [no.] of the list.

7.5 Examples

Example 1:

Version update of a EASYMEN file with appendix. Appendix unchanged.

Start Appman or reset with [Clear] .
Load new extension file with option
[Load][Extension(+Appendix)].
Load old extension file with appendix with the same option.

Now all definitions are listed.

Save new extension file with the option
[Save][Extension(+Appendix)].
The old file can be overwritten.

Example 2:

A menu definition has been modified and shall be exchanged in the appendix.

Start Appman or reset with [Clear] .
Load extension file with appendix with the option.
[Load][Extension(+Appendix)].
Load new menu with the option
[Load][Menu [_men]].
Select old menu in column [type + no.] and type 0 (zero).
Select new menu in column [type + no.] and type position.
Select new menu in column [name] and type name.
Save extension file with the option
[Save][Extension(+Appendix)].
The old file can be overwritten directly.

Example 3:

The appendix of an old EASYMEN and an old EASYPTR shall be copied to a new 'ptrmen_cde' file which contains all SuperBASIC extensions in one (shorter) file.

Start Appman or reset with [Clear].
Load new ptrmen_cde file with the option
[Load][Extension(+Appendix)].
Load old EASYPTR extension file with appendix with
[Load][Extension(+Appendix)].
Load old EASYMEN extension file with appendix with
[Load][Extension(+Appendix)] too.
The complete appendix is now listed.

Save new extension file with the option
[Save][Extension(+Appendix)].

The new file should be given a new name.

8.0 The SuperBASIC extensions

EASYBMEN	file 'easybmen_cde'
EASYPTR	file 'easyptr_cde'
also contained in	file 'easyptrr_cde'
	file 'ptrmen_cde'
	file 'ptrmenr_cde'
EASYMEN	file 'easymen_cde'
also contained in	file 'easymenr_cde'
	file 'ptrmen_cde'
	file 'ptrmenr_cde'
partly contained in (see → chapter 12)	file 'easybmen_cde'
PUTGET	file 'putget_cde'

EASYPTR helps you developing a menu structure for your program as quickly and easily as possible. With the SuperBASIC extensions you are given routines to draw and control the menu in your program. Naturally you must write the program itself and connect it with the menu. In general a SELect ON loop is the best way to do this. Here the MCALL function value can be used directly to select which procedures and functions shall be called.
You may study the demo programs. The programs should run normally, but these are only examples of how to use our SuperBASIC extensions. EASYPTR is not intended to be a programming lesson, so we assume that you are familiar with SuperBASIC principles.

The Pointer Environment Kit tutorial disk will provide a better introduction to programming with EASYPTR and we recommend that you also study the examples on that disk.

If you are not familiar with the pointer environment features, please read Appendix I before you start.

8.1 Preliminary remarks

Basics

The EASYPTR extensions give access to the extended system routines of the pointer interface. Most commands are directly based on a Trap #3 operation.
The EASYMEN extensions can control menu definitions. Most commands are directly based on vectored routines of the window manager. These read the data structures and use the Trap #3 system routines of the pointer interface and the standard screen driver as required.

A tip to start with:

Load the file 'ptrmenr_cde' residently (i.e. use LRESPR) and work with it. Later you can decide which type of extension file with appended definitions is best for your program. Use the EASYEXT extensions and the PTRMEN_EXT command to load and unload extension files with appended definitions temporarily.

Working area

The EASYPTR commands RDPT, SPRS and WMOV and the EASYMEN commands MSETUP and MDRAW, install a menu working area for the window channel. This area is linked into the 'thing' list and can be seen in the QPAC2 'Things menu'. The name is always 'easymenu' together with the full QDOS channel ID in hex, e.g. easymenu001B0015 . The areas installed on the EASYMEN level are compatible with those on the EASYPTR level, i.e. all EASYPTR commands (RDPT, WMOV ...) use the same working area as installed by MDRAW. On the other hand, if a menu working area is installed, then the pointer working area will automatically be removed.
Working areas installed on the menu level are removed with MCLEAR, which removes the menu and restores the screen contents under the menu. Working areas on the pointer level are removed with

CLPT. All working areas of a job can be removed with CLAMP at once. But here the window contents is not restored. This command is intended to be of some use during program development.

Why different files ?

The extensions are available in different files. Some applications may only use the menu level extensions, then it is sufficient to have the EASYMEN commands in one file. Other applications may built up their own window and/or menu control with the EASYPTR commands, then it is sufficient to have these in one file. If a program works on both levels, the PTRMEN file should be taken, as it is about 2Kb shorter than the single files together.

The EASYBMEN commands (chapter 12) are identical with those of the same name in the EASYMEN extensions. Even more, the EASYBMEN file has all commands, except the MAW... commands for dynamic application sub-window menus, though not described in chapter 12.

The full syntax can be used with all commands in the EASYBMEN file! So, if a program does not need dynamic application sub-window menus, the shorter EASYBMEN extension file can be used.

If you intend to run several programs, you should load the extension residently, instead of blowing up memory with multiple copies. Here you can use the 'r' versions, which have some additional commands useful during program development. These are the commands PTR_EXT and MEN_EXT (PTRMEN_EXT), SPRITES and MENUS (APPENDIX) and MLIST.

You can link your menu and sprite-definitions in the compiled code anyway. APPMAN offers a special file format (app) where a single command APPAn gives access to appended definitions.

File	Contents	Purpose
easyptr_cde	pointer commands	pointer interface access
easyptrr_cde		**resident version**
	pointer commands	pointer interface access
		program development
		usage by several programs
easybmen_cde	menu commands	menu control without dynamic application sub-window menus
easymen_cde	menu commands	menu control
easymenr_cde		**resident version**
		menu commands menu control
		program development
		usage by several programs
ptrmen_cde	pointer commands	pointer interface access
	menu commands	menu control
ptrmenr_cde		**resident version**
		pointer commands
		pointer interface access
	menu commands	menu control
		program development
		usage by several programs
putget_cde	string commands	direct memory string read/write

8.2 Basic ideas

Primary window

The first window channel opened for a job. In SuperBASIC (JOB 0) this is channel #0. The primary windows of all jobs are managed by the pointer interface. Switching between jobs is switching between their primary windows. The interface automatically saves and restores the window contents within primary channel outline when jobs are switched around. All subsequently opened window channels are called 'secondary windows'.

Error handling

All EASYBMEN commands offer an error handling which is especially useful in compiled programs. If the parameter for the channel number is an integer and passed as a name (e.g. ch%), then if an error occurred the procedure will return and the error code is passed in the variable ch%, else the variable is left unchanged.
Examples. 100 MDRAW #0
or 100 MDRAW #ch
will return an error in the standard way, (the program would stop and the error would be reported), but
100 ch%=0:RDPT #ch%
will always return with no error on the program level and the program itself may test on an error, e.g.:
110 IF ch%>0:MYERROR ch%
where MYERROR might be a special error handling procedure within the program.

Hierarchy

All well behaved (see → Outline) window channels of a job form a hierarchical structure. The pointer interface scans all secondary windows in reverse order (the latest one is on top) and only the latest one declared to be well behaved (outline set) will get a pointer request.
A pointer request on the primary window is only possible if there is no well behaved secondary window.
A pointer request is not possible on a window if the outline has not been set.
The commands OUTL and CLPT (part II only) .offer the flexibility to manage several window channels together with the pointer interface, though if all features of the pointer environment together with EASYPTR are used it will be possible to work with a single window channel (see → EASYMENU see → menu elements see → MWINDOW) and to open secondary windows only for temporary purposes (e.g. pop-down menus). This will avoid hierarchical problems.

Outline

The area within which a pointer request can be started on a window is called the outline. The outline is set with the command (see → OUTL, part II only). A pointer read is only possible when the outline has been set. MDRAW sets the outline automatically.
All Standard I/O-commands are only active in the WINDOW area of a window.
The WINDOW size of a channel may be equal to or smaller than the outline size.
The outline of a primary channel may cover the whole screen, while a secondary window outline always has to be within its primary window outline.
A channel is called well behaved, if the outline has been set.
When a menu is drawn with MDRAW, the outline is set to the menu main window size. If the channel is a primary window, all secondary windows which are open will be forced to be within the menu window.
When a menu is drawn in a secondary window, and if the primary window outline is not set, then it is forced to the minimum line which includes all actually open secondary windows. A menu for a secondary window can only be drawn within its primaries outline, i.e. the outline must be large enough.
The window size is often changed by the window manager when a menu is drawn and controlled. The application too may set the window size for its purposes at will within the outline (main window size). If the menu structure has been destroyed, an MDRAW may restore it.

Pointer Request

A pointer request is only possible on the latest opened window channel of a job for which an outline has been set (Hierarchy).

Secondary Window

All windows opened after the primary channel of a job are called secondary windows.

8.3 Syntax

The syntax description follows the usual conventions.
All parameters in square brackets […] can be omitted.
Parameters in curly brackets {…} are alternative parameters, i.e. one of these parameters (mostly of different types) must be given.
All parameters which do not have square brackets elsewhere around them, must be given.
For all variables and parameters the SuperBASIC conventions about SuperBASIC names, parameter types (floating point, integer (%) and string ($)) , etc. are valid.
Names can be given as SuperBASIC names (type floating point) or string expressions (between inverted commas), where nothing else is mentioned. In general we recommend to use string expressions in programs, to keep the SuperBASIC name table free from such things.
This applies to all parameters where **'name'** is a part of the variable name in the syntax description (Name can be name$ too).

The EASYBMEN commands are identical to those of the EASYMEN extensions. If you have purchased EASYPTR part I (Base Version) only you will have only short descriptions of the commands. This manual contains to the complete descriptions . If you are not familiar with programming in SuperBASIC you may wish, as an exercise, to experiment with the shorter descriptions. These can be found as an appendix t this manual. The full syntax is supposed to be of use even for very complex programs and thereby more difficult to read. The best way to get to grips with programming using EASTPTR is to read the text files and try the examples on the 'PEK' disk which you should have received with this manual. If you did not receive this disk contact Q Branch.

8.4 Channel number

The channel number can be replaced by the address of the working definition, as soon as the working area is installed (see → MWDEF). By this the routines are about twice as fast This is especially significant in MCALL or RDPT loops.

The channel number is always defaulted by the primary windows channel number. But this leads to a significant slow down and should only be used during program development.

8.5 Commands

For any command the file(s) were it can be found is given. In order to find command descriptions fast, the following syntax description is in alphabetical order, although sometimes it would be better to describe. correlated commands next to each other. All commands do have references to related commands.

WARNING

Some Qliberator versions have problems with a separator
, \ ! TO
at the end of a parameter list (system crashes). Set a 'dummy parameter', e.g. \0 behind it or look for a new version of Qliberator.

9.0 Keywords, Description and Syntax

APPAn

in **EASYAPP files**
see → **SPRA**

adr = APPAn (name$)

n	can be 0 - 9, depending on which number has been set in APPMAN when the file was created.
name$	name of appended definition (always string!)

Returns the address of the definition in an EASYAPP file
The compiler directive to include EASYAPP files with Qliberator is

```
REMark $$asmb=?,0,64
```

APPENDIX

in **ptrmenr_cde**
see → **MENUS, SPRITES**

APPENDIX **{#ch%}{device}**

ch%	SuperBASIC channel number default: #1
device	device or filename

A list of all definitions (menus, sprites, blobs, patterns) appended to the actually loaded PTRMEN extensions is printed to the given channel.

Example

'ptrmenr_cde' is loaded residently. A copy of 'ptrmen_cde' renamed 'mymen_cde' with appended definitions is loaded with XJ during program development. The command APPENDIX, which is only present in the resident version, finds the actual loaded appendix of the PTRMEN file automatically and prints the list.

APPENDIX	gives the list in #1
APPENDIX ser1	will print it to the printer
APPENDIX 'flp1_appendixlist'	will write to this file.

BLOBW

in	**easyptr_cde** **easyptrr_cde** **ptrmen_cde** **ptrmenr_cde**
see →	**SPRW**
BLOBW	**[#ch%,] xpos%, ypos%, blob, pattern »»** **»» {[, spray%]}{[, endxpos%, endypos%]}**
ch%	SuperBASIC channel number (see → 8.4) default: primary channel of the job
xpos%	x-pixel (start-) position (relative to window origin)
ypos%	y-pixel (start-) position (relative to window origin)
blob	address, or name of appended blob definition
pattern	address, or name of appended pattern definition
spray%	number of pixels to be sprayed
endxpos%	x-pixel end position
endypos%	y-pixel end position
1. Variant	BLOBW [#ch%,] xpos%, ypos%, blob, pattern

Write a blob filled with the pattern at the given position

2. Variant BLOBW [#ch%,] xpos%, ypos%, blob, pattern, spray

Spray 'spray%' pixels of the given pattern in the blob at the given position. The pixels are randomly sprayed. Several calls may fill the blob, but it is not sure that the entire blob is ever filled.

3. Variant BLOBW [#ch%,] xpos%, ypos%, blob, pattern, endxpos%, endypos%

A line of blobs filled with the pattern is drawn from the start to the end position.

Remark: Any sprite definition can be used as a blob (i.e. only the pattern mask of all non-transparent pixels of the sprite definition is used). Any sprite definition whose x-size is a multiple of 16 can be used as a pattern.

CLAMP

in	**easyptr_cde** **easyptrr_cde** **ptrmen_cde** **ptrmenr_cde**
see →	**CLPT, MCLEAR**
CLAMP	**[jobID]**
jobid	QDOS Job ID = job tag*65536 + job number → Toolkit 2 → NXJOB default: calling job

Clear all menu and pointer working areas of a job. All memory reserved for the working areas is released. All window channels (except the primary window) are reset to the be unmanaged.

If jobID is not the calling job, then the job will be removed too (except job 0).

CLPT

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde
see → **CLAMP, MCLEAR**

CLPT **[#ch%]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job.

Clear pointer working area installed by RDPT, SPRS or WMOV of a window channel. All memory reserved for the working area is released. The window channel is reset to the unmanaged (not well behaved) stage.

FLIM

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde

FLIM **[#ch%,] xsz%, ysz%, xor%, yor%**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

xsz%
ysz%
xor%
yor% return parameters for size and origin

Find the maximum window limits, where the windows outline can be set. For primary channels this is the screen size, for secondary channels the primary outline.

GET$

in **putget_cde**
see → **PUT$**

text$ = GET$ (adr)

text$ string found

adr address in memory

Reads a standard QDOS string directly from a given address. The address must be even and pointing to the length word.

L_WSA

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde
see → **S_WSA, WSAIN, WSASV, WSARS**

adr = L_WSA (filename[, xsz%, ysz%])

adr function value = address of the save area

filename name or string

xsz%
ysz% return parameter. x/y-pixel size of loaded screen save area.

Memory in the size of the file length is reserved on the common heap and the file is loaded there. This area can then be managed with WSARS, WSASV and S_WSA.

The memory can be released with
RECHP adr
or CLCHP.

The contents of the loaded file is only checked so far that, if the two return parameters are present, it looks for the flag $4AFC. Without the two return parameters for the size, the command is equivalent to

```
adr=ALCHP(FLEN(filename))
LBYTES filename, adr
```


MAWBAR

in **easymen_cde**
easymenr_cde
ptrmen_cde
ptrmenr_cde
see → **MAWDRAW, MAWCLEAR**

MAWBAR **[#ch%]{,}{\} num, wi%, he%[, xst%, yst%[, xra%, yra%]]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

separator , in MCALL: return after key press
\ in MCALL: immediate return

num application sub-window number

wi% number of virtual columns
<0 → a split is possible

he% number of virtual rows
<0 → a split is possible

xst% start column
default: 0

yst% start row
default: 0

xra% grid width 0-255
default: 6 pixels

yra% grid height 0-255
default: 10 pixels

Set up and draw application sub-window bars. The window contents is not changed. The length of the bar corresponds to the number of visible grid elements. The control and positioning must be made by the calling program. The separator before 'num' decides, whether MCALL returns immediately when the pointer is moved into the application sub-window or only when a key is pressed. This flag can be set with MAWDRAW too.

A special menu item number (upper word of the function value) is returned by MCALL if bars are present. This number can be evaluated with MAWNUM.

MAWBARR

in **easymen_cde**
easymenr_cde
ptrmen_cde
ptrmenr_cde
see → **MAWBAR**

MAWBARR **[#ch%]{,}{\} num, wi%, he%[, xst%, yst%[, xra%, yra%]]**

This command is the same as MAWBAR, but the scroll/pan arrows are also shown. An even more extended special menu item number is returned in MCALL if the arrows are clicked.

MAWCLEAR

in	**easymen_cde** **easymenr_cde** **ptrmen_cde** **ptrmenr_cde**
see →	**MAWBAR, MAWDRAW, MAWSETUP**

MAWCLEAR **[#ch%,] num**

ch%	SuperBASIC channel number (see → 8.4) default: primary channel of the job
num	application sub-window number, e.g. MCALL function value

Clear application sub-window menu previously set up With MAWDRAW, or bars previously set up with MAWBAR.

MAWDRAW

in	**easymen_cde** **easymenr_cde** **ptrmen_cde** **ptrmenr_cde**
see →	**MAWSETUP, MAWCLEAR**

MAWDRAW **[#ch%,] num[, array$]**

ch%	SuperBASIC channel number (see → 8.4) default: primary channel of the job
num	application sub-window number (low word). MCALL function value can be used directly.
Array$	2- or 3-dimensional string array default: existing menu

Set up application sub-window menu. The menu is based on a 2- or 3- dimensional string array. All data is evaluated from the array size. If the menu does not fit into the window size, scroll or pan arrows and bars are set up automatically. The menu then can be split up to 4 times. The command also redraws an existing menu, the variable array$ is ignored then.

Complete syntax for advanced users:

MAWDRAW **[#ch%] {,}{;}{\} num[, [array$][, xst%, yst%[, [under%] »»**
»» [, [xsz%], [ysz%][, [xju%][, [yju%]]]]]]

separator
, MCALL return if an item is hit.
\ MCALL also returns after a scroll or pan operation
; MCALL returns after scroll and pan operations when the pointer is moved next.

xst% start column

yst% start row

under% number of the character to be underlined. This character is also set to be the selection key. Can be the same value for all items or set separately for any item if under% is a 1- or 2-dimensional integer array.
The dimension(s) must match the corresponding dimension(s) of array$.
default: no underline

xsz% column width. Can be a single value, then all columns have the same width, or a 1-dimensional integer array with values for each column.
The dimension must match the first dimension of array$.
default: max. text length (last dimension of array$) optimized to window width

ysz% row height. Can be a single value, then all rows have the same height, or a 1-dimensional integer array with values for each row. The dimension must match the first dimension of array$ if array$ is 2-dimensional or the second if array$ is 3-dimensional.

xju% object x-justification
0 centered
> 0 distance in pixels from the left
< 0 distance in pixels from the right
Can be a single value, then the justification is the same for all items, or a 1- or 2-dimensional integer array with values for each item.
The dimension must match the corresponding dimensions of array$.

yju% object y-justification
0 centered
> 0 distance in pixels from the top
< 0 distance in pixels from the bottom
Can be a single value, then the justification is the same for all items, or a 1- or 2-dimensional integer array with values for each item.
The dimension must match the corresponding dimensions of array$.

With the additional parameters a menu structure can be set up individually configured. The parameters from 'under%' can be omitted. If there are more parameters, only the comma must be present.

Example: MAWDRAW #3,1,array$,0,0,,,16 with 16 pixel high rows.

Specialty: the elements in array$ can be the address of objects. This is signaled by the two characters @»

.
Example: array$(3,4)='@ »'&adr

The object type is passed with the parameter 'under%':

under%	= 0	text object, no underline
	> 0	text object with underline
	= -2	sprite object
	= -4	blob object
	= -6	pattern object

Blob and pattern objects require a combine partner to be set. This is done in the ATTRIBUTES menu in EASYMENU. If there is no combine partner, then a bad parameter error will be reported.

MAWDRAW is also used to redraw an existing menu. Then only the x/y start values take effect, all others are ignored and can be omitted.

Example: MAWDRAW #3,num„3,7
will redraw the menu with column 3, row 7 in the top-left corner. 'num' can be the MCALL function value, only the application subwindow number is used.

With the separator before num several return options are available. E.g., the option 'return after scroll or pan operation' could be used to monitor a value without selecting it.
The return option with the separator flag can be reset with any MAWDRAW call. 4n return after a scroll or pan operation, MCALL will only return the application sub-window number. The menu status can be evaluated with MAWNUM.

The MAWSETUP syntax is identical with MAWDRAW-The only difference is that MAWDRAW after having set up the menu structure will draw the menu immediately, while MAWSETUP will return. It is then possible to set the item status, which initially is available for all items, with MSTAT or MSTAT%. Then the menu can be drawn with MAWDRAW in it's initial state, thus avoiding unpleasant flickering effects.

MAWITEM

in **easymen_cde**
easymenr_cde
ptrmen_cde
ptrmenr_cde
see → **MINOB, MITEM**

MAWITEM **[#ch%,] num, [under%], {text$}{name}{adr}[, key$]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

num item number (e.g. MCALL function value)

under% object type
= 0 text, no underline
> 0 text, number of character to be underlined
= -2 sprite
= -4 blob
= -6 pattern
default: 0

alternative
text$ character string
name name of appended definition
adr address of object

key$ new selection key.
default: old selection key, or underline character

A new object is set for an existing application sub-window menu item. With MAWITEM and MTEXT$ the objects of an application sub-window menu can be controlled like an array. A menu can be empty when set up with MAWDRAW and filled as required with MINPUT.

REMARK: on menus set up with MAWDRAW, the text length may not exceed the last dimension size of the string array. On static menus set up in EASYMENU new text objects can only be set with the address option. ATTENTION Blob and pattern objects require a combine partner to be set. This is done in the ATTRIBUTES menu in EASYMENU. If there is no combine partner, then a bad parameter error will be reported.

Specialty: If text objects are set with the address, either in the string array of MAWDRAW or once with MAWITEM, they can be edited with MINPUT directly at the address.
Then they does not need to be set again with MAWITEM.

MAWNUM

in **easymen_cde**
easybmen_cde
easymenr_cde
ptrmen_cde
ptrmenr_cde
see → **MCALL**

minum = MAWNUM ([#ch%]{,}(\} awnum[, xpos%, ypos%])

minum	function value: pure menu item number 0 is returned if awnum is not the item selected last or awnum is not an application sub-window item number (e.g. loose item)			
ch%	SuperBASIC channel number (see → 8.4) default: primary channel of the job			
separator	,	evaluation of the last selected item		
	\	evaluation of the top left item (start column/row)		
awnum	call:	separator	,	menu item number (see → MCALL)
			\	application sub-window number
	return:			application sub-window number
xpos%, ypos%	call: return parameter, must be passed by name return: column/row number the columns/rows are numbered from 0. these values refer directly to the element of the string array if the menu was set up with MAWDRAW.			

Evaluation of the special menu item number returned by MCALL for application sub-window menus. Either the last selected item or the actual position (e.g., to get the start column/row after pan/scroll operations) of the menu is evaluated.

MAWS

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde

MAWS **[#ch%]{,}{\} acc%, wup%**

ch%	SuperBASIC channel number (see → 8.4) default: primary channel of the job	
separator	,	set
	\	read
acc%	mouse acceleration values: 0 - 9 default: 9	
wup%	mouse wake up speed values: 0 - 9 default: 2	

Set or read mouse speed. The values take global effect, i.e. all programs are affected. Therefore, if a program needs altering the speed, it should reset to the old values afterwards.

MAWSETUP

in **easymen_cde**
easymenr_cde
ptrmen_cde
ptrmenr_cde
see → **MAWCLEAR, MAWDRAW**

The syntax is identical with MAWDRAW. The only difference is that with MAWSETUP the menu is not drawn immediately.

MCALL

in **easymen_cde**
easybmen_cde
easymenr_cde
ptrmen_cde
ptrmenr_cde

num = MCALL ([#ch%][{,}{\}{!} num1, stat1%[, ...]])

num — function value menu item number or application sub-window number or special item number
see below!

ch% — SuperBASIC channel number (see → 8.4)
default: primary channel of the job

separator
- , return:
  - if a menu item (loose or application menu) is hit,
  - if the pointer is moved into an application sub-window without menu
- \ too, if an event occurred. Then a special item number is returned (see below!)
- ! the selection key is returned instead of the item number

num1 — menu item number

stat1% — new item status
- 0 set to available
- 1 set to selected
- -1 set to unavailable

num2 — ...

stat2% — ...

Menu Control Parameters

The status of menu items can be controlled by passing parameter pairs (num1, stat1%, ...). The items are drawn in the new status. There is no restriction on number and order of the pairs. The item number can be 0 or a pure application sub-window number, then all items which have the change flag set will be redrawn (see → MSTAT, MSTAT%)

Function value:

The control of the menu driven program is done on evaluating the MCALL function value. This refers to the menu item number or a special item number for certain events:

range **menu item or event**

-1088,-1025

event	**key**	**special item number**
action	DO	-1025 = -1024 - 1
cancel	ESC	-1026 = -1024 - 2
help	F1	-1028 = -1024 - 4
move	CF4	-1032 = -1024 - 8
re-size	CF3	-1040 = -1024 -16
sleep	CF1	-1056 = -1024 - 32
wake	CF2	-1088 = -1024 - 64

-1024,-1 negative loose menu item number

0 menu was removed

1,256 application sub-window number

>65536 application sub-window menu item number. All items are numbered row-wise from top left to bottom right beginning with 1. If 'minum' is this item number and 'awnum' is the number of the application sub-window, then the function value is: **minum*65536+awnum**
This value can be evaluated with MAWNUM.

If there are bars set up with MAWBAR in the application sub-window, then a special operation code is delivered. The lower word is again the application sub-window number, but the upper word is special:

Bit 0	Section 1
Bit 1	Join sections
Bit 2	Pan bar
Bit 3	Do
Bits 4-15	Pixel position

The evaluation can be directly done using masks

```
op_code = minum && 15
pix_pos = INT(minum / 16)
```

or by giving MAWNUM two additional (normally used for the x/y start position) parameters. minum is then the pure operation code, x_st% is the pixel position and y_st% is the size of the scroll/pan bar.

Possible operation codes are:

0	HIT on scroll bar in section 0
1	HIT on scroll bar in section 1 (if divided)
8	DO on scroll bar
10	DO on scroll bar in section 0 (join)
11	DO on scroll bar in section 1 (join)
4	HIT on pan bar in section 0
5	HIT on pan bar in section 1 (if divided)
12	DO on pan bar
14	DO on pan bar in section 0 (join)
15	DO on pan bar in section 1 (join)

If there are also arrows (menu set up using MAWBARR) then a slightly different word can be returned as well:

Bit 0	always 0
Bit 1	action on up/down arrows
Bit 2	action on left/right arrows
Bit 3	DO
Bit 4-15	always 1

Accordingly, if pix_pos is -1 the possible values for op_code are:

0	HIT on up
2	HIT on down
4	HIT on left
6	HIT on right
8	DO on up
10	DO on down
12	DO on left
14	DO on right

MCALLT

in **easymen_cde**
easybmen_cde
easymenr_cde
ptrmen_cde
ptrmenr_cde
see → **MCALL**

num = MCALLT ([#ch%,] event%, time% [, num1, stat1%[, ...]])

event%	job and window event vector. Must be given as a name
time%	timeout in 1/50th of a second after which MCALLT should return
num1	menu item number
stat1%	new item status 0 set to available 1 set to selected -1 set to unavailable
num2	...
stat2%	...

Event% and time% are obligatory, i.e. they must be present. Event% must be given as a name, not only as value, as the event which caused the return will be returned in event%. Event% can be used to pass both window events and job events to force a return. The job events are masked into the upper byte of event%.
The function value will be -1280 if MCALLT returned because of an event or timeout. In case of a timeout evenr% will be set to 0 (since Basic extension 4.09).

event% = event to return
= win_events + job_events%256
time% = timeout to return

Note: if you want MCALLT to return on any event like MCALL with the "\" separator then you have to set event% to 255 before calling MCALLT!

MCALLT only works with SMSQ/E from v2.84 or WMAN from v1.52.
Attention: MCALLT does not make any version tests!

MCLEAR

in	**easymen_cde** **easybmen_cde** **easymenr_cde** **ptrmen_cde** **ptrmenr_cde**
see →	**CLAMP, CLPT, MDRAW, MSETUP**

MCLEAR **[#ch%]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

An existing menu is removed. The window contents below the menu is restored. Secondary windows are reset to be unmanaged.

MDRAW

in	**easymen_cde** **easybmen_cde** **easymenr_cde** **ptrmen_cde** **ptrmenr_cde**
see →	**MCLEAR, MSETUP**

MDRAW **[#ch%,][{name}{filename}{adr}][, xpos%, ypos%[, xsz%, ysz%]]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

name name of an appended menu definition

filename single menu definition file with ending '_men'

adr address of a definition (see → APPAn)

xpos%
ypos% x-/y pixel origin of the menu
default: dynamic: actual pointer position
fixed: as set in EASYMENU

xsz%
ysz% x-/y size
A menu definition can have several layouts. The window manager searches automatically for a layout which accommodates the given size. The given size also determines the scaling factor for scalable elements. See chapter 4.3 for more details on this subject.

The working definition (thing) is set up and the menu is drawn. The window channel must be opened before. An existing menu is redrawn. The positioning of the window is either relative to the actual pointer position or fixed, according to the selection made in EASYMENU. This selection is overwritten if an origin position is given with MDRAW. The origin of the main window is always set to a x-position dividable by 4 and an even y-position.

MSETUP

has the same syntax as MDRAW. The only difference is that the menu is not drawn immediately.

MENUS

in **easymenr_cde**
see → **APPENDIX, SPRITES**

MENUS **{#ch%}{devicename}**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

devicename file or device which shall be opened for writing and closed afterwards.

A list of the appendix (menus, sprites, blobs, patterns) of the actually initialized EASYMEN extensions is printed to the given channel, file or device.

MEN_EXT

in **easymenr_cde**
see → **PTR_EXT, PTRMEN_EXT**

MEN_EXT

The commands in the resident version of the EASYMEN extension become valid again.

Example: 'easymenr_cde' is loaded residently. During program development a version of the EASYMEN extensions with appended definitions is loaded temporarily with XJ. These are removed with RXJ after programming is finished. Now MEN_EXT will reset the commands.

MINOB
in **easymen_cde**
easymenr_cde
ptrmen_cde
ptrmenr_cde
see → **MAWITEM, MITEM**

MINOB **[#ch%][{,}{\}]{info%,inob%}{num! under%}, {text$}{name}{adr}**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

separator , redraw window
\ don't redraw window (see → MLIDRAW)

alternative
info% info window number

inob% info object number

or:

num = inob%*65536+info% followed by'!' separator

under% object type
= 0 text, no underline
> 0 text, number of the character to undeline
= -2 sprite
= -4 blob
= -6 pattern
default: 0 text, no underline

alternative

text$	character string
name	name of appended definition
adr	address of object

A new object is set for an info sub-window object.

Blob and pattern objects require a combine partner to be set. This is done in the ATTRIBUTES menu in EASYMENU. If there is no combine partner, then a bad parameter error will be reported.

Specialty: If text objects are set with the address once with MINOB, they can be edited with MINPUT directly at the address. Then they must not be set again with MINOB, but only redrawn with MIWDRAW.

MINPUT

in	**easymen_cde** **easybmen_cde** **easymenr_cde** **ptrmen_cde** **ptrmenr_cde**	
MINPUT	**[#ch%,]{text$}{adr}[\]**	
ch%	SuperBASIC channel number (see → 8.4) default: primary channel of the job	
alternative	text$ adr	name of a string variable! address of string (see → GETS, PUTS)
separator	none \	cursor is at the end cursor is at the start

MINPUT only works on a window when a pointer or menu working definition exists

(see → MDRAW, RDPT, SPRS, WMOV).

MINPUT is terminated with:

ESC	text$ unchanged
ENTER	new text is accepted
Cursor up or down	new text is accepted

The key can be evaluated with MKEY% or PVAL (result%(6)).

The text string given in 'text$' or at 'adr' is edited. The variable text$ must be passed as a name.

For specialists only:

The two commands GET$ and PUT$ allow to read or write standard QDOS strings directly at a given address in memory. Such an address can be passed to MINPUT directly. The string is then edited there. This is very fast, saves memory and does not affect the SuperBASIC variable area. Text objects set in menu items or info objects with their address must not be reset with MAWITEM, MINOB or MITEM, but must only be redrawn (see → MAWDRAW, MIWDRAW, MLIDRAW, MSTAT, MSTAT% or MCALL).
Be aware that MINPUT needs a buffer of the following size:

Buffer size: Number of characters which fit into the window width from the actual cursor position +1 byte.

MITEM

in **easymen_cde**
easymenr _cde
ptrmen_cde
ptrmenr_cde
see → **MAWITEM, MINOB**

MITEM **[#ch%,] num, [under%], {text$}{name}{adr}[, key$]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

num item number (e.g. MCALL function value)

under% object type

= 0	text, no underlined
> 0	text, number of character to be underlined
= -2	sprite
= -4	blob
= -6	pattern

default: 0

alternative

text$ character string

name name of appended definition

adr address of object

key$ new selection key.
default: old selection key, or underline character

A new object is set for a loose menu item.

ATTENTION

Blob and pattern objects require a combine partner to be, set. This is done in the ATTRIBUTES menu in EASYMENU. If there is no combine partner, then a bad parameter error will be reported.

Specialty:

If text objects are set with the address once with MITEM, they can be edited with MINPUT directly at the address.

Then they don't need to be set again with MITEM.

MIWDRAW

in **easymen_cde**
easymenr_cde
ptrmen_cde
ptrmenr_cde
see → **MAWDRAW, MDRAW, MLIDRAW**

MIWDRAW **[#ch%]{, num}{\ bit mask}**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

alternative: , num info window number
\ bit mask long word bit mask

For specialists only:

With the bit mask parameter the first 32 info windows can be redrawn at once. A window is redrawn, if the corresponding bit in the 32-bit longword mask is clear. The bit numbers start from 0.

Example:

$FFFFFFFF	=	-1	all bits set, no window is redrawn
$FFFFFFFE	=	-2	bit 0 clear, info window 1 is redrawn
$FFFFFFF5	=	-10	bit 1 and bit 3 clear, info windows 2 and 4 are redrawn

MKEY%

in **easymen_cde**
easybmen_cde
easymenr_cde
ptrmen_cde
ptrmenr_cde
see → **MCALL**

key% = MKEY% ([#ch%])

ch% SuperBASIC channel number (see – 8.4)
default: primary channel of the job

key% ASCII code The key which caused return from MCALL, MINPUT or RDPT.

MLIDRAW

in **easymen_cde**
easymenr_cde
ptrmen_cde
ptrmenr_cde
see → **MAWDRAW, MDRAW, MIWDRAW**

MLIDRAW **[#ch%,] [num]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

num loose item number (MCALL function value)
default: **all loose items**

The loose item is, or all loose items are redrawn.

MLIST

in **easymenr_cde**
ptrmenr_cde

MLIST **[{#ch%}{device},]{#menu_ch%}{menu_adr}**

alternative: ch% output channel
default: #1
device output file or device

alternative: menu_ch% menu window channel number
menu_adr menu working definition address (see → MWDEF)

A list of all menu elements of an existing menu is printed to the given channel. The list is mainly intended to be a reference for the item and window numbers.

Example: in #3 a menu is drawn with MDRAW:
MLIST flp2_menulist,#3
or MLIST par,#3
or MUST #3

MRDIM

in **easymen_cde**
easymenr_cde
ptrmen_cde
ptrmenr_cde
see → **MSTAT**

MRDIM **[#ch%,] num, xdim%, ydim%**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

num ≤ 0 loose items and info windows
> 0 application sub-window number

xdim% loose menu items dimension or x-dimension of application sub-window menu

ydim% info windows dimension or y-dimension of application sub-window menu

The dimension is *number of elements -1*. E.g., to be used for MSTAT.

MREMB

In **easymen_cde**
easymenr_cde
ptrmen_cde
ptrmenr_cde
see → **MCALL, MINPUT**

MREMB **[#ch%]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

The current item border is removed.

MSETUP

in **easymen_cde**
easymenr_cde
ptrmen_cde
ptrmenr_cde
see → **MCLEAR, MDRAW**

The MSETUP syntax is identical to MDRAW.

The only difference is that MSETUP does not draw the menu.
After the working definition has been installed with MSETUP, the initial item status can be set (see → MSTAT or MSTAT%) avoiding unpleasant flickering effects.
Then the menu is drawn with MDRAW. It is also possible to draw only parts of the definition (see → MAWDRAW, MLIDRAW or MIWDRAW).

MSTAT

in **easymen_cde**
easybmen_cde
easymenr_cde
ptrmen_cde
ptrmenr_cde
see → **MCALL, MDRAW, MSTAT%**

MSTAT [#ch%,]{TO}{\} number, stat%

ch%	SuperBASIC channel number (see → 8.4) default: primary channel of the job	
separator	,	copy array stat% to status block, no redraw
	TO	copy array stat% to status block and redraw
	\	copy status block to array stat%
number	≤ 0	loose menu items
	> 0	items in application sub-window
stat%	integer array. The dimension must be the number of items less one (as DIM starts counting from 0). Use MRDIM!	

With MSTAT the menu item status (unavailable,selected and available) of all loose items or of all items in an application sub-window menu can be controlled with an integer array.

Here it is easy to control the status of interdependent items, or to read which items of a list actually are selected.

In 'number' the MCALL function value can be passed directly, only the relevant information (negative for loose items or application sub-window number) is used.

MSTAT reads (separator \), writes (separator,), or writes and redraws (separator TO) the status:

ATTENTION: If the status is set only, **a command to redraw the status must follow**. Otherwise those items which have the redraw flag set will be unavailable independent of the status set.
A redraw command can be a MCALL with a status parameter pair, MDRAW or MAWDRAW, MSTAT% or another MSTAT with the redraw option set.

Values in stat%:

READ:	0	available
	16	unavailable
	128	selected
WRITE:	1	set to available
	17	set to unavailable
	129	set to selected

The value 1 (bit 0 set) added to status value is the redraw flag, which signals the draw routines which items shall be redrawn.

If a MDRAW or MAWDRAW is used to draw the whole menu, then the redraw flag can be omitted, i.e. only 0, 16 or 128 is set.

In the “Simplified syntax” chapter 12 an example of the use of MSTAT can be found.

MSTAT%

in	**easymen_cde** **easybmen_cde** **easymenr_cde** **ptrmen_cde** **ptrmenr_cde**
see →	**MCALL, MDRAW, MSTAT**

stat%= MSTAT% ([#ch%,] number[{,}(TO} newstat%[\]])

ch%	SuperBASIC channel number (see → 8.4) default: primary channel of the job
stat%	actual item status 0 available 1 selected -1 unavailable -7 menu item does not exist
number	menu item number (MCALL function value)
newstat%	status to be set.

Reads (without newstat%) or writes and sets the status of an item. The item is only redrawn if the status has been changed (separator,) or if redraw is forced with a TO separator.

A \ separator after 'newstat%' will cause that the item is not redrawn. This option can be used if several items are set to a new status. Then only the redraw flag is set and all items changed can be redrawn at once.
MSTAT% with the redraw option enabled (no \ at the end) will redraw all items which have the redraw flag set.
Thus it is sufficient to enable redrawing with the last MSTAT% of a list for several items.

ATTENTION: If the status is set only, a command to redraw the item must follow. Otherwise those items which have the redraw flag set will be unavailable independent of the status set.
A redraw command can be a MCALL with a status parameter pair, MDRAW or MAWDRAW, MSTAT or another MSTAT% with the redraw option enabled.

ALL option: If number = 0, then all loose items of the main window are set to the given status. If 'number' is the pure application sub-window number, then all menu items of an application sub-window menu are set to the given status.

MTEXT$

in **easymen_cde**
easymenr_cde
ptrmen_cde
ptrmenr_cde

text$ = MTEXT$ ([#ch%,] [{num}{info%, inob%}])

text$	function return value
ch%	SuperBASIC channel number (see → 8.4) default: primary channel of the job

alternative:

num	menu item number < 0 loose menu items > 0 application sub-window menu items

or:

info%	info window number
inob%	info object number

The text object of an item or object is returned ('***' if the object is no text).
If only the channel number is passed, the menu name is returned. This can be used to test the existence of a menu.

MOBJA

in **easymen_cde**
easymenr_cde
ptrmen_cde
ptrmenr_cde

adr = MOBJA ([#ch%{,}{\}] {num}{info%!}{info%, inob})

ch%	SuperBASIC channel number (see → 8.4) default: primary channel of the job
separator	, get address of the object's text/sprite \ get address of the object's definition block in the working definition

alternative:

num	menu item number < 0 loose menu items > 0 application sub-window menu items

or:

info%!	info window number (only applicable if separator is \)

or:

info%	info window number
inob%	info object number

This function returns the address of an object set in a loose menu item or info object (Text, Sprite...). With this one can for example reuse sprites of menus in other parts of the program without having to attach or load them again.

For specialists only:

With the \ separator MOBJA returns the address of the object's definition block. Of course it's also possible to work through all definitions on your own using MWDEF.

MWDEF

in **easymen_cde**
easymenr_cde
ptrmen_cde
ptrmenr_cde

adr = MWDEF ([#ch%])

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

The address of a pointer or menu working definition is returned. This address can afterwards always be used instead of the channel number. The'#' must be there anyway! The commands are then executed about twice as fast, as searching the working definition is not necessary. But, the calling program is responsible that there is the working definition at the given address! The structure is not tested by any routine.

Example:
```
ch=FOPEN('con'): MDRAW#ch,mymenu:ch%=ch
adr=MWDEF(#ch%)
action=MCALL(#adr)
if ch%<0 THEN ERROR_ROUTINE:ch%=ch
```

Error handling

If MWDEF was called with an integer variable 'ch%', then this variable is saved in the working definition. If an error occurs subsequently in a routine called with the address of the working definition, then the error is returned in 'ch%'. This option can be switched on (call with integer variable) or off (call with floating point variable) with MWDEF calls. This error handling is overwritten if a command is called with an integer channel number '#ch%' (see → 7.2, error handling)

Remark:
The error variable must be reset to a non-negative value after an error occurred!

Example: ...
```
ch=FOPEN('con')
MDRAW#ch,mymenu
ch%=ch
adr=MWDEF(#ch%)
action=MCALL(#adr)
if ch%<O:ERRORROUTINE:ch%=ch
```

MWINDOW

in **easymen_cde**
easybmen_cde
easymenr_cde
ptrmen_cde
ptrmenr_cde

MWINDOW **[#ch%{,}{\}] num[!]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

separator
- , set window colours
- \ don't alter window settings

num
- < 0 loose menu item
- > 0 application sub-window number
  or
  menu item number (MCALL function value) of the last selected item in an application sub-window menu.
  or
  info window number followed by separator '!'
- 0 main window outline

The window area is set to the size of the menu element. All standard I/O commands then take effect in this area. If a semicolon is used as separator, then the colours and character settings of the channel remain untouched. Otherwise the following rules apply:

- all character settings reset to OVER 0, CSIZE 0,0
- loose item colours corresponding to an available loose item are set
- application window paper colour of application window is used with white ink
- information window paper colour of information window is used with white ink
- main window paper colour of main window is used with main window system palette ink $202

MWLINK

in **easymen_cde**
easymenr_cde
ptrmen_cde
ptrmenr_cde

MWLINK **[#ch%{,}{\}] num{,}{!} #link_ch%**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

separator
- , set window colours
- \ don't alter window settings

num as for MWINDOW link ch% channel number of the channel to be linked.

The window size of 'link-ch% is set to the size of the menu element. See MWINDOW for a description of how the colours are handled.

The command only works if the outline is not set for 'link_ch%'.

OUTL

in	**easyptr_cde** **easyptrr_cde** **ptrmen_cde** **ptrmenr_cde**
OUTL	**[#ch%,] [xsz%, ysz%, xor%, yor%[, xsh%, ysh%[, fl]]]**
ch%	SuperBASIC channel number (see → 8.4) default: primary channel of the job
xsz% ysz%	x-/y- pixel size -1/-1 keep size unchanged default: primary window: the smallest line which includes all actually open window channels of the job secondary window: actual window size
xor% yor%	x /y-pixel origin default: primary window: origin of the smallest line (see above) secondary window: actual window origin
xsh% ysh%	x-/y-pixel shadow size default: 0 = no shadow
fl	flag: 0 or absent keep contents 1 don't keep contents default: 0

The outline of a window channel is set (see → 7.2). The window is set to be managed by the pointer device (pointer requests are only possible on managed windows!).

If OUTL is used with parameters, then the screen contents behind the outline is automatically saved (see → WRES). OUTL without parameters (e.g. OUTL #2) does not save the screen contents!

OUTL on a primary channel moves all secondary channels, if the size is unchanged (this is checked internally, not only if the size is passed as -1,-1).

OUTL is not possible if there is an existing working definition for the channel installed with MDRAW, RDPT, SPRS or WMOV.

While MDRAW sets the outline automatically, it is absolutely necessary to set the outline for the channel and its primary channel before RDPT,...

Examples:

```
OUTL
```

will set the outline of the jobs primary window to the smallest line which includes all actually open windows of the job.

```
OUTL #2
```

will set the outline of #2 to the actual window size. The screen background is not saved!

```
OUTL #2,256,100,50,50
```

will set the outline to the given size and origin. The screen background is saved. If the actual window size and origin is not within the outline, it is set to the outline size.

```
WINDOW #2,70,50,80,60
```

will set the window size. The window size now is restricted always to be within the outline.

```
OUTL #2,-1,-1,100,90
```

will move the whole channel to the new position. The window contents is unchanged, also the window will have been moved to the origin position 130,100.

PINF

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde

PINF **[#ch%]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

If there is no pointer environment present, an error will be reported.

PTOP

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde

PTOP **[#ch%,] {jobnum}{jobID}{jobname}[{,}{\}][text$]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

jobnum job number as found with JOBS or QJ

jobID jobtag*65536+jobnum

jobname full or part of jobs name. The first job found of which the name begins with 'jobname' is taken.

Separator , pick only
\ pick and wake

text$ character string

The job is picked to the top. If the \ flag is present, a wake event is sent too.
If a text parameter is present, then text$ is inserted into the keyboard queue of the job.

ATTENTION:

The channel number can be omitted, then the primary channel is taken which must exist, i.e. the procedure does not work from within a job which does not have an open window channel!

Example: PTOP fil\

Will pick the first 'Files' job (QPAC2) and sends a wake, i.e. the directory is renewed.
PTOP 'text', FILE SELECT$
might pick 'text87', call the FILE SELECT$ menu (menu extensions) and write the filename into the keyboard queue of text87.

PTRMEN_EXT
PTR_EXT

in **easyptrr_cde**
ptrmenr_cde
see → **MEN_EXT**

PTRMEN_EXT
PTR_EXT

The commands in the resident versions 'ptrmenr_cde' or 'easyptrr_cde' become valid again (see MEN_EXT explanations).

PUT$

in **putget_cde**

see → **GET$**

PUT$** **adr, text$

A standard QDOS string (2 bytes length followed by the character bytes) is written to the given address in memory.
PUT$ and GET$ allow fast control of string objects in menu elements. MINPUT allows direct access to such strings in memory.

PVAL

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde
see → **MCALL, MINPUT, MKEY%, RUPT**

PVAL **[#ch%,] result%**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

result% return parameter: integer array for the result of a pointer request.
The dimension must be at least 15
i.e. → DIM result%(15)

Gives the result of a pointer request (see → RDPT or see → MCALL) in an integer array:
result%(0) = channel tag (see → channel ID)
result%(1) = channel number
result%(2) = application sub-window number
result%(3) = pointer x-position (relative)
result%(4) = pointer y-position (relative)
result%(5) = mouse button down: 1 HIT, 2 DO
result%(6) = ASCII code of key pressed
result%(7) = termination vector (see → RDPT)
result%(8) = window width
result%(9) = height
result%(10) = x-origin
result%(11) = y-origin
result%(12) = call channel tag
result%(13) = call channel number
result%(14) = pointer x-position (absolute)
result%(15) = pointer y-position (absolute)

optional, if the dimension is at least 16:

result%(16) = event vector

Explanations:

result%(0) and result%(1)
give the QDOS channel ID of the channel the pointer actually is in.

result%(2)
is the application sub-window number in which the pointer actually is in. A value of -1 indicates that the pointer is in no application sub-window. Numbering here starts at 0, while numbering of application sub-windows for the EASYMEN commands MCALL, MWINDOW,... starts at 1. So add 1 to get the EASYMEN number of the application sub-window.

result%(3) and result%(4)
give the pointer position relative to the window origin in result%(10) and result%(11).

result%(5)
signals that a mouse button or SPACE / ENTER is held down.

result%(8) to result%(11)
give the coordinates of the window the pointer is in.

result%(12) and result%(13)
give the QDOS channel ID of the calling window.

result%(14) and result%(15)
give the pointer position in absolute screen coordinates.

result%(16)
gives the event

value	key	event
1	DO	do action
2	ESC	cancel
4	F1	help
8	CF4	move
16	CF3	change size
32	CF1	sleep
64	CF2	wake

RDPT

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde

RDPT **[#ch%,] [{,}{\} [tvec%[, xpos%, ypos%[, time%]]]]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

separator , ignore window events
\ return on window events

tvec% call parameter: termination vector
default: 1 = key press
return parameter: high byte key code
low byte unchanged

xpos%
ypos% pointer position
default: old position
call parameter:
xpos%>0, ypos%>0 absolute position
xpos%<0, ypos%>0 position relative window origin
xpos%>0, ypos%<0 position relative outline origin
return parameter:
absolute pointer position

time% timeout
default: -1 = infinite

Start pointer request. The routine returns if the pointer event given in the termination vector occurred, or when a finite timeout elapsed.

Attention:

A pointer request is only possible if the outline (see → OUTL) has been set for the channel and its primary channel!

Termination vector:

The following pointer events can cause a return, multiple events are possible (add values):

Value	Event
1	key stroke
2	key down
4	key u p
8	pointer moved
16	pointer out of window
32	pointer in window

The key code is returned in the high byte of tvec%: keycode=INT(tvec%/256)
tvec% must not be changed for the next call, as only the low byte is used to determine the termination vector!

Pointer position:

The pointer is set to the position given in xpos%, ypos%, but not if the position is unchanged. The values can be passed negative, this is used to determine the absolute position. This new position (not the previous one) is used to decide whether the pointer has been moved.

…RDPT

Return parameters:

A fast evaluation of the pointer request is possible through the key code delivered with tvec% and the new absolute pointer position in xpos%, ypos% (naturally the variables must have been passed by name for this). This can make a separate call to PVAL unnecessary.

Immediate return

A call with tvec%=48 or time%=0 will return immediately. This can be used to set the pointer position or to install the working area.

Examples: OUTL:RDPT
will bring the pointer to the screen. On a key press the request is terminated.
OUTL #1
RDPT #1,32,20,20
will bring the pointer at the absolute position 20,20. If this position is within window #1, an immediate return will follow, else the pointer must be moved into #1 to cause a return.

Remark:
To remove the working areas for #0 and #1, enter
CLAMP
or CLPT#O:CLPT#1

REMPT

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde

REMPT **[#ch%]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

Removes the pointer (sprite) from the screen. For graphical applications which must access the area under the pointer.

RPXL%

RPXL%

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde

ncol% = RPXL% ([#ch%,] [xpix%, ypix%[, key%, col%]])

ncol%	function value:	pixel colour found -1 window limit reached
ch%	SuperBASIC channel number (see → 8.4) default: primary channel of the job	
xpix% ypix%	call parameter: start position relative to window origin return parameter (only if key% is present): x-/y-position found. If ncol%=-1 and xpix% or ypix%=-1, the window limit has been reached before the searched colour was found.	
key%	scan key: -1 read pixel colour or sum up: 0 reference colour is in xpix%/ypix% 1 reference colour is in col% 0 search up 2 search down 4 search left 6 search right 0 search different colour 8 search same colour	
col%	reference colour	

Returns the pixel colour at the given position or scans according to key in the given direction until the reference colour is found.

Note: SMSQ/E in high colour mode does currently not support this call. This is no EASYPTR limitation.

S_WSA

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde
see → **L_WSA, WSAIN, WSASY, WSARS**

S_WSA **[#ch%,] adr, filename**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

adr address of area installed with WSAIN or loaded with L_WSA

filename to save the area

The part window save area is saved on a file. The file is in the pointer interface format for part window save areas. This file can be loaded with the
[Cut out] [QL Pointer IF]
option in EASYSPRITE if the ending is '_pic'.

The file can be reloaded with L_WSA.

SPRA

in **easyptrr_cde**
ptrmenr_cde
see → **APPAn**

adr = SPRA ({name}{number})

adr function value: address of definition

name name,

number or number of appended definition

The address of a sprite-, blob- or pattern definition appended to an EASYPTR extension file is returned.

SPRITES

in **easyptrr_cde**
see → **MENUS, APPENDIX**

SPRITES **[{#ch%}{device}]**

ch% output channel
default: #1

device output file or device

A list of all appended sprite, blob and pattern definition is printed to the given file.

SPRS

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde

SPRS **[#ch%,] [{names1}{adr1}][, {name2}{adr2}]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

name1
adr1 name of an appended (or address of) definition. This is set to be the pointer sprite within the windows outline.
0 = standard arrow
default: 0

name2
adr2 ditto for the window area.
default: outline pointer sprite

Set the pointer sprite for the outline and the window area. If there is no working definition, it will be installed automatically by SPRS.

SPRW

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde
see → **BLOBW**

SPRW **[#ch%,] xpos%, ypos%, {name}{adr}**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

xpos%
ypos% x-/y-pixel position relative to window origin

name name of an appended or

adr address of a sprite definition.
-1 to -37 pointer interface internal sprites.

The sprite is written with its origin at the given position.

WMOV

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde
see → **OUTL**

WMOV **[#ch%,] [{-1}{xpos%, ypos%}]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

xpos%
ypos% move window to this position

If no position is given, the move window routine is started. The window move sprite is used to position the window.
All secondary windows of a primary window are moved simultaneously.

If -1 is given as the sole parameter, then the WMAN internal move routine will be used, which allows for opaque/outline moves in latest WMAN releases. In this case secondary windows will NOT be moved along.

WRES

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde
see → **WSAV**

WRES **[#ch%][{\}{,1}]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

separator \

or flag ,1 → keep save area.

The screen contents saved with WSAV or automatically with OUTL or MDRAW is restored.

Examples:
Make menu invisible temporarily

```
MDRAW#3, mymenu
...
WRES#3\              :REMark restore background
...
MDRAW#3              :REMark redraw menu
```

Restore background of a temporarily used window

```
ch=FOPEN('con')
OUTL#ch,100,100,50,50
...
RDPT #ch,...
...
WRES#ch              :REMark restore background
CLPT#ch              :REMark remove working area
CLOSE#ch
```


WSAIN

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde
see → **L_WSA, S_WSA, WSASV, WARS**

adr = WSAIN ([#ch%,] [xsz%, ysz%])

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

xsz%
ysz% x-/y-size of the area default: window hit size (outline-shadow)

Install a part window save area. The area can be larger than the physical screen size. The only restriction is the available memory size.

Remark:
WSAIN only installs the save area and sets up the appropriate header. Save and restore of window parts is done with WSASV and WSARS. The save area can be released with RECHP adr or CLCH P or automatically with WSARS. S_WSA can be used to save the area to a file.

For specialists:
The area is reserved on the common heap and has the following format

adr-20	longword	heap length
adr-16	longword	driver
adr-12	longword	owner job
adr-4	longword	ALCHP link
adr	word	flag $4AFC
adr+2	word	x-pixel size
adr+4	word	y-pixel size
adr+6	word	line increment
adr+8	byte	screen mode
adr+10	bytes	screen image

WSASV

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde
see → **L_WSA, S_WSA, WSAIN, WSARS**

WSASV **[#ch%,] adr[, xsz%, ysz%, xor%, yor%[, xst%, yst%]]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

adr address of save area installed with WSAIN or loaded with L_WSA

xsz%
ysz% x /y-pixel size of area to be saved default: outline size

xor%
yor% x-/y-pixel origin of area to be saved relative to window origin default: outline origin

xst%
yst% x-/y-pixel start position in save area

Saves the whole screen contents within the outline, or a part of the screen within the window area (set with WINDOW or MWINDOW).

Difference to WSAV:
As WSAV the variant of WSASV without additional parameters
(e.g. WSASV #3,adr)

saves the screen contents within the outline. But, as WSAV uses the internal pointer interface save area, WSASV uses a separate area.

Also the WSAV area is not in a suitable form to be saved on a file.

WSARS

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde
see → **L_WSA, S_WSA, WSAIN, WSASV**

WSARS **[#ch%,] adr [[{,}{\}] [, xsz%, ysz%, xor%, yor% [, xst%, yst%]]]**

ch% SuperBASIC channel number (see – 8.4)
default: primary channel of the job

adr address of save area installed with WSAIN or loaded with L_WSA

separator , don't keep save area
\ keep save area

xsz%
ysz% x-/y-pixel size of area to be restored default: outline size

xor%
yor% x-/y-pixel origin of area to be restored relative to window origin default: outline origin

xst%
yst% x-/y-pixel start position in save area

Restores the outline screen contents (e.g. WSARS#3,adr\) or a part of the window contents from the save area.
The separator behind 'adr' can be used to keep or give up the save area.

The save area can also be released with
RECHP adr
or CLCHP
On the EASYPTR SuperBASIC Disk an utility program can be found, which demonstrates the usage of WSAIN, WSASV, WSARS and S_WSA. If the program is executed, screen cuts can be made, which are suitable to be loaded with the [Cut out] option in EASYSPRITE. Thus sprites can be cut out from any screen.

WSAV

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde
see → **WRES**

WSAV **[#ch%]**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

Saves the screen contents within the outline of a window channel. Uses the internal save area of the pointer interface. The same area is used, when the screen contents is saved automatically with MDRAW, OUTL or WMOV. This contents is overwritten by WSAV! To avoid this, use see → WSASV instead.

WSIZE

in **easyptr_cde**
easyptrr_cde
ptrmen_cde
ptrmenr_cde

WSIZE **[#ch%,] xpos%, ypos%**

ch% SuperBASIC channel number (see → 8.4)
default: primary channel of the job

xpos%
ypos% how much the user moved the resize mouse pointer

Shows a standard window resize sprite and, after the user clicks or hits a key, returns the difference from the original position. This can then be added/subtracted (depending on which window corner your resize button resides) from the current window size to get the new window size. See chapter 4.3 for more details.

10.0 Final remarks

Don't get confused by the many features of the SuperBASIC extensions. Most commands can be used with a few parameters. Test them in the interpreter by just typing the command.

Look at the syntax definition very carefully. Only those parameters which are not between square brackets have to be given. Where flags can be passed with a separator, normally the comma is the standard form. Non-present parameters are conveniently defaulted, if necessary at all.

The existence of the many optional parameters and flags allow very complex programs to be written. It is best to begin with the, easily understandable, EASYBMEN syntax.

Use commands and optional parameters only if your program needs them. In this way you might find it easier to understand the structure of the parameters and flags.

Developing the user interface of a program often takes most of the developing time. EASYPTR may help you to save a lot of time. Please don't expect that it can all be done in a few seconds. Developing a menu with EASYMENU and sprites with EASYSPRITE for a program will also take some time.

With EASYPTR you get tools which you probably could not find for other operation systems, or if they were available, you would have to pay a lot of money for them.

The English manual is a translation from the original German manual. It was done by myself (with some amendment by Roy Wood of Q Branch) . If you find any mistakes or omissions please contact Q Branch with details so that we can put the right.

Please keep in mind that you belong to a small minority amongst QL enthusiasts, program developers working with the pointer environment. You can, therefore, imagine that the sales figures for this program package are rather moderate. Please support the community of programmers and vendors by not making illegal copies of this software and manual and passing them on to other users. On the other hand if you use the package and find it useful or write a distributed program by using it please let other people know.

The thing we need more than anything else in the QDOS/SMSQ scene is more programs!

If you are working seriously with EASYPTR, you should check for a updates regularly. With an update of EASYPTR you always get also the latest versions of the pointer environment and the menu extensions.

11.0 The Pointer Environment

The Pointer Environment is a system extension to QDOS. Since it appeared first with QRAM it has been further developed and enhanced. Use always the latest versions. We recommend the use of SMSQ/E which incorporates the latest version of the Pointer Environment with other, newer, features such as event handling, and, on systems which support them, extended screen sizes and colour drivers. SMSQ/E is available for most QL platforms and is an integral part of QPC2 the PC Windows QL emulator. For further details contact Q Branch or Jochen Merz Software.

The pointer environment consists of three parts:

Pointer interface file 'ptr_gen'
Window manager file 'wman'
Hotkey system file 'hot_rext'
All three parts must be loaded residently when the computer is ‘booted’ They should be loaded in the following order. If Toolkit 2 and Lighting are used, they must be initialized first:

```
100 lngINIT
110 TK2_EXT
120 LRESPR 'flp1_ptr_gen'
130 LRESPR 'flp1_wman'
140 LRESPR 'flp1_hot_rext'
```

Then other extensions can be loaded in undefined order, e.g.

```
150 LRESPR 'flp1_qpac2'
160 LRESPR 'flp1_easyext_res'
170 LRESPR 'flp1_menu_rext'
```

Then the commands to install hotkeys and the button frame could follow. Also the SuperBASIC extensions EASYBMEN or PTRMEN (full version only) could follow, if you intend to use them permanently.

```
300 LRESPR 'flp1_easybmen_cde'
or
300 LRESPR 'flp1_ptrmenr_cde'
```

11.1 Pointer Interface

The Pointer Interface is a system extension to QDOS. It installs the pointing device which makes it possible to move a pointer on the screen. Generally this is a small item (sprite.) of max. 64 x 48 pixels. This sprite may be moved with either a mouse or the cursor keys. Thus

a menu for program control may be managed,
a pencil in a paint program may be moved,
objects may be marked and moved around etc.

The pointer interface can be controlled by the keyboard alone, but a mouse gives more flexibility and ease of use once the user has become familiar with it. For the standard QL / Aurora there are two hardware mice packages:

QIMI Mouse Interface
superHermes (available from T.F. Services)

and the software SERmouse serial mouse driver. (available from Q Branch and Jochen Merz Software). Under SERmouse the mouse may even simulate the cursor keys when moved while the left button is pressed. Thus e.g. it may be used in any Editor (ED, QUILL etc.).

The Pointer Interface includes two SuperBASIC extensions CKEYON and CKEYOFF:

11.1.1 CKEYON

The mouse pointer can be moved with the cursor keys.

11.1.2 CKEYOFF

The mouse pointer can not be moved with the cursor keys.

Multitasking with the pointer interface

QDOS allows several programs to run at the same time (multi-tasking). But standard QDOS has no option to save the window contents of a program (destroyable windows). Each program must have a separate option to restore its windows. CTRL C is used to switch around the keyboard queues, but the window contents is not restored and the program on top destroys the windows of other programs.

The pointer interface - by help of an extended channel definition - saves and restores the window contents automatically when the programs are switched around. This extended channel definition is the reason why software which does not use the system routines correctly (e.g. direct manipulations in the channel definition blocks) does not run under the pointer environment.
QJUMPs QRAM or QPAC 2 offer several ways to adapt badly behaved software (especially the PSION suite). But all serious software suppliers now offer program versions that run with the pointer environment. (unfortunately some of them don't use it.)

Programs (jobs) are switched around with CTRL C as with standard QDOS, but now the pointer interface switches around the pile of primary windows that belong to every job and no longer the simple keyboard queue.

The Pointer Interface distinguishes between two different types of windows of a job.

1. The primary window is the first window channel opened for this job.
2. All subsequent opened window channels are called secondary windows.

The channel definition not only keeps the window size but also the window outline which is equal to or beyond the window size area. Pointer requests are restricted to the outline area while all standard input/output is restricted to the window size area.

The primary window outline may cover the whole screen. A secondary window is restricted to the area within that primary outline.

To keep the new window definition compatible with standard QDOS, window channels are managed by the Pointer Interface in two different ways.
In the case of unmanaged windows (in general windows of those programs which are not written for the Pointer Interface) it watches automatically that the primaries outline always covers all window channel areas opened for this program.
When a primary window is marked to be managed (also called well behaved, by setting the outline explicitly see → **OUTL**) then the program itself has to make sure that all its windows are within its primary windows outline. Pointer requests are only possible on managed windows. All managed windows of a program (job) form a hierarchical structure. The Pointer Interface allows pointer requests only to the latest window declared to be managed.

11.2 Programmer access

In the pointer interface there are some system routines implemented as

```
TRAP #3          $6C-$7F
```

to which the assembler programmer has access. the technical description is in the **QPTR** manual.
The SuperBASIC extensions of EASYPTR (part II only) give access to the system routines of the pointer interface for the SuperBASIC programmer.

11.3 Window Manager

11.3.1 Control elements

The window manager offers standardized elements to control basic function of a menu.

11.3.2 Events

The following keys are connected with standardized events:

key	**event**
SPACE	select
HIT (left mouse key)	select
ENTER	do action
DO (right mouse key)	do action
ESC	cancel
F1	help
CTRL F1	sleep
CTRL F2	wake
CTRL F3	change window size
CTRL F4	move window

11.3.3 Window Manager Programmers Access

The technical description of the window manager routines is to be found in the **QPTR** manual. The complexity of these routines and the menu structure to be controlled by them, were the main reasons that EASYPTR was first written.

With the menu generator EASYMENU a window definition in the window manager standard can be designed on the screen.
The SuperBASIC extensions EASYBMEN (base version) or EASYMEN (part II) give anybody, who is familiar with the basics of SuperBASIC programming, the opportunity to handle such menus from a SuperBASIC program

Part II - (SuperBASIC) only

Assembler programmers can avoid setting up and debugging endless data structures, as EASYSOURCE delivers the fully commented source code of a menu structure designed with EASYMENU. Simple applications can be realized by just adding the pointers to action routines.

12.0 Simplified syntax

The following short descriptions of the commands are those included with Part 1 of the EASYPTR toolkit in its original release. Now that we have collated Parts 1 and 2 as a single disk we have placed the full syntax of the commands in the main body of the text. These descriptions are, however, of use to those who want to 'ease gently' in using this powerful package. As mentioned before we recommend that you also obtain and study the Pointer Environment Kit – a P/D disk of examples and tutorials – which will help beginners to get to grips with the concepts behind this package and use the full facilities that are available.

MAWNUM

num = MAWNUM ([#ch%,] number, row%, column%)

Call parameters:

ch%	SuperBASIC channel number default: the primary channel
number	**MCALL** function return value
row%	undefined, must be passed by name
column%	undefined, must be passed by name

Return parameters:

num	item number All items are numbered (starting with 1) from top left to bottom right. 0 see → number was not an application sub-window item/window number.
row%	item row number (starting from 0)
column%	item column number (starting from 0)

This function is used to evaluate the special application sub-window menu/window number which is delivered by MCALL if an item in an application sub-window is hit.

MCALL

num = MCALL (#ch%)

ch%	SuperBASIC channel number default: the primary channel
num	Result of the menu request. < 0 negative loose menu item number = 0 menu removed > 0 application sub-window number > 65536 special item/window number of application sub-window menu evaluation with MAWNUM

The loose menu item number and the application sub-window number are the same as in column [No.] in the ELEMENT LIST menu in EASYMENU.

The function to start a menu request. The function returns if:
- a menu item is selected
- the pointer has moved into an application sub-window which does not have a menu.

Exceptions:

Selection keys code 3, 5 and 7

Three actions are controlled by MCALL internally:

If a loose menu item has selection key ASCII code 3 , then the menu will be removed if this item is selected (implicit MCLEAR). MCALL will return with 0. If you want to proceed a simple cancel action, set the selection key to ESC (ASCII code 27) and the function will return normally.

If a loose menu item has selection key ASCII code 5, then a move window action is processed, if this item is selected.

If a loose item has selection key ASCII code 7, then the job will be made a sleep button if the QPAC2 button frame is present.

MCLEAR

MCLEAR [#ch%]

ch%	SuperBASIC channel number default: the primary channel

A menu, previously drawn with MDRAW is removed. All memory reserved for the menu working definition is released. **The channel is not closed.**

MDRAW

MDRAW **[#ch%,] [{name}{filename}]**

ch%	SuperBASIC channel number default: the primary channel
name	name of an appended menu definition
filename	file name of a menu definition

Loads or searches the menu definition. The working definition is set up and the menu drawn.
If there is an existing menu for this channel, it is redrawn.

MINPUT

MINPUT **[#ch%,] instring$ [\0]**

ch%	SuperBASIC channel number default: the primary channel
instring$	string variable. **Must always be passed by name**

A string may be edited in the given channel. instring$ must always be passed as a name and may initially be an unset or empty string variable. ENTER and Cursor up or down will be accepted as termination. ESC will cancel the input (instring$ is unchanged). The termination key can be evaluated with MKEY%. By default, the cursor is at the end of the string. If there is a \0 at the end, then the cursor will be at the first character of the string.
MINPUT is only possible on a menu window.

Example:
Edit a filename with suggested directory

```
400 file$= 'flp2_'
410 MINPUT #3,file$
420 OPEN#4,file$
…
```

MKEY%

key% = MKEY% ([ch%])

ch%	SuperBASIC channel number default: the primary channel
key%	the ASCII code of the key which caused MINPUT or MCALL to return, e.g.: 1 = HIT 2 = DO 3 = ESC ...

To decide, whether an item was hit by **HIT** or **DO**.

MSTAT

MSTAT	**[#ch%,] {TO}{\}number, stat%**	
ch%	SuperBASIC channel number default: the primary channel	
separator	TO \	copy array stat% to status block and redraw copy status block to array stat%
number	≤ 0 > 0	loose menu items items in application sub-window
stat%	integer array. The dimension must be the number of items less one (as DIM starts counting from 0).	

With MSTAT and MSTAT% the menu item status (unavailable, selected and available) can be controlled.
While MSTAT% works on a single item, MSTAT allows to control the status of all loose items or all items of an application sub-window menu at once. Here it is easy to control the status of interdependent items, or to read which items of a list actually are selected.
In 'number' the MCALL function value can be passed directly, only the relevant information (negative for loose items or application sub-window number) is used.

MSTAT reads (separator \) or writes (separator TO) the status:

Values in stat%:

READ:	0 available
	16 unavailable
	128 selected
WRITE:	1 set to available
	17 set to unavailable
	129 set to selected

Example:

Menu with 8 loose items, where 6,7 and 8 shall deselect each other. One application sub-window with menu (1 column, 15 rows):

```
...
130 DIM stat%(7),awstat%(14)
...
160 MDRAW#3,'flp2 test men'
170 stat%(6-1)=129                  :REMark 6 selected
180 oldon=6                         :REMark save
190 MSTAT#3 TO 0,stat%
...
200 REPeat loop
210 number=MCALL(#3)
220 SELect ON number
230 =0:STOP
240 =-5 TO -1:stat%(-number-1)=1:MPACT number
220 =-8 TO -6:stat%(oldon)=1:stat%(-number-1)=129
230 oldon=-number
240 =REMAINDER:
250   num=MAWNUM(#3,number,row%,column%)
255   IF NOT num:NEXT loop
260   key%=MKEY%(#3)
270   IF key%>1
280     AWACT
290     FOR n=0 to 14:awstat%(n)=1
300     MSTAT#3 TO 1,awstat%
305   END IF
310 END SELect
320 MSTAT#3 TO 0,stat%
330 END REPeat loop
...
400 DEFine PROCedure MPACT(number)
...
500 END DEFine
...
600 DEFine PROCedure AWACT
...
700 END DEFine
...
```


MSTAT%

stat% = MSTAT% ([#ch%,] number [, newstat%])

ch%	SuperBASIC channel number default: the primary channel
stat%	actual item status 0 available 1 selected -1 unavailable
number	menu item number (MCALL function value)
newstat%	status to be set.

Reads (without newstat%) or writes and sets the status of an item.

Example:

```
...
160 MDRAW#3,'flp2_test_men'
...
200 REPeat loop
210 number=MCALL(#3)
220 SELect ON number
...
280   = -3:stat%=MSTAT%(#3,number,0)
...
```

MWINDOW

MWINDOW	**[#ch%,] number[!]**
ch%	SuperBASIC channel number default: the primary channel
number	negative loose menu item number or application sub-window number or Information sub-window number followed by a '!' separator.

The numbers are the same as in column [No.] in the {Loose items...} {Infowindow...)... LIST menu in EASYMENU.

The window size of the channel is set to the size of the menu element. The MCALL function value can be passed in 'number' directly. All standard SuperBASIC commands (PAPER,INK, CLS, CSIZE, INPUT, PRINT,..) work on this area afterwards, as if a WINDOW command would have been used. MDRAW will restore the menu structure at any time.

13.0 Command index

