**EASYPTR 3**

**Development System**

**for the**

**Pointer Environment**

**Part III**

**Albin Hessler Software**

**© 1992-93 Albin Hessler Software**

**Contents Part III**

**EASYPTR 3 Library Routines**

**III.0 EASYMENU/EASYSPRITE changes** 159

III.0.1 Menu element and subwindow numbering 160

**III.1 The Source Code Generator EASYSOURCE** 161

III.1.1 Configuration 161

III.1.2 Menu Items 162

III.1.3 EASYSOURCE 3.01 165

**III.2 Pointer Interface Library Routines** 167

III.2.0 List of pointer interface routines 167

III.2.1 How the functions work 168

III.2.2 Error codes 169

III.2.3 Tips and tricks for C beginners 170

III.2.4 Timeout 171

III.2.5 QDOS channel ID 171

III.2.6 Primary window, secondary window, outline 171

III.2.7 Assembler 171

III.2.8 Symbols used for call and return parameters 171

III.2.9 The routines 171

(alphabetical order)

**III.3 The window manager Library Routines** 199

III.3.0 List of window manager routines 200

III.3.1 Internal routines 202

III.3.2 The routines 204

(alphabetical order)

**III.4 The EASY MENu Control Library Routines** 247

III.4.0 List of EASYMENC Routines 248

III.4.1 Internal routines 249

III.4.2 Register A5 and A6 249

III.4.3 Address pointers 249

III.4.4 Parameter passing (assembler only) 250

III.4.5 Action routines 252

III.4.6 Action routines written in C 252

III.4.7 Primary window 253

III.4.8 The routines 254

(alphabetical order)

**III.5 Data structures** 285

III.5.0 Result area 285

III.5.1 Menu window definition 287

III.5.2 Window working definition 297

III.5.3 Window status area 307

III.5.4 Extended channel definition structure 311

III.5.5 EASYMENC extended working definition 312

III.5.6 EASYMENCasm library routines overview 312

**111.0 EASYMENU/EASYSPRITE changes   
S-ROFF**Both jobs now have an option to create a relocatable object tile from a saved menu definition or from a single sprite/blob/pattern definition.

EASYMENU   
Select S-ROFF in the Files menu

EASYSPRITE   
Select S-ROFF in the Save as menu.

The relocatable files can be used directly to link the definitions into the object code with the linker.

The definitions are given an external label which is the name of the definition as defined in the Files menu.

The module name is derived from the filename.   
The section name is MDATA.

As menu definitions generated from EASYMENU do not contain pointers to action-, hit-, draw- and control routines, the usage of relocatable menu definition modules makes only sense together with EM\_SETUP.

Within a C module which shall use the definition, the label must be declared to be external.   
As long as the structure of a definition must not be accessed from within a C program the type does not matter, as only the address must be known.

Example:

extern long testmenu;

then

&testmenu is the address.

**0.1** **Menu element and sub-window numbering**While in EASYMENU and for the SuperBASIC commands all menu item and sub-window numbers start from one, the window manager itself start numbering from zero.   
As the assembler and C programmer may acces the menu structures directly, all library routines also expect that the numbering starts from zero.

**111.1 The Source Code Generator**

**EASYSOURCE** File ‘easysource\_exe’

EASYSOURCE is a special disassembler for menu and sprite definitions as generated with EASYMENU or EASYSPRITE. The disassembly is fully commented. Even more, EASYSOUPCE disassembles any correct menu definition, it it is preceeded by the EASYMENU header as described below. By this, definitions written by hand can be transformed in a fully commented disassembly.   
The resulting source code listing may be used directly in assembler programs. To control menus, the technical guide QPTR is needed.   
Simple applications can be realized without deep knowledge of the QPTR technical resources. EASYSOURCE offers two options to generate the complete code necessary to make a job around the menu, or to make it a pulldown menu. Menu items like [Move] or [ESC] then already are active, the other items must be given an action routine only.

**1.1 Configuration**EASYSOURCE can be configurated with ‘config’. The assembler choice can be set to ‘GST’ or other’.   
If you choose ‘GST’, the resulting source code is ready to be assembled as a link module.   
When ‘other’ has been choosen, all labels are suffixed with the definition name. Thus several definitions may be in the same code.   
If ‘GST’ is your assembler, then you should take this option, else a listing disassembled in both modes may help you to make the best choice for your assembler program.

**1.2 Menu Items**

The menu is in a pure text form and is widely self explanatory.

**[menu]**

A menu definition ist translated. The result is a commented source code listing. The comments are the same as in the QPTR technical guide. Sprite definitions included as objects in the menu definition are also fully disassembled. Any correct menu definition is accepted as input, if it has the EASYMENU header, where only the word menu’ is significant.

easymenu\_header

dc.I 0,0   
dc.b ‘menu’   
dc.l 0,0,0,0

menu

**[sprite]**

Translates a sprite definition. Any correct sprite definition is accepted. The definitions generated with EASYSPRITE (ending ‘\_spr’) are pure definitions.

**[EASYMENU header]**

If this selection is on, the EASYMENU header is written out with the menu definition in the listing. This reassembled (probably modified) definition will be loaded in EASYMENU without any problem.

**ATTENTION:**Pointers to action routines will be lost if loaded in EASYMENU.

**[job]**

Together with the menu definition the complete code to make a self standing executable program Gob) is written out. You have only to assemble it. Actions like move window’ and ‘ESC’ will work, if the appropriate selection codes (5 or 3) where set.

**action routines**All other menu items can be activated by setting the action routine pointer:

…

dc.w action1-\* ;pointer to action routine

…

The routine must control the status of the menu item. To set the item available again, the supplied routine ‘unselect’ may be used.   
Action routines must return with

…

moveq #0,d4   
moveq #0,d0   
rts

if the menu request shall continue. To redraw the item status, WM.LDRAW must be called. For this purpose the supplied routine ‘Imi\_draw’ can be used. If d4 contains an event number, or if dOoO or the condition codes are set, then WM.RPTR returns. The routine ‘action1’ could look somewhat like this:

action1

bsr unselect   
… ;your routine   
bsr lmi\_draw   
moveq #0,d4   
moveq #0,d0

rts

**Application sub-window routines**Together with the menu code a set of standard application sub-window ‘setup’, draw, ‘hit’ and ‘control’ routines is always written out if there is an application sub-window in the definition. For windows set up with the ‘bars only’ option the program must take it’s control within the control routine.   
The routines are designed for standard purposes and may be modified as required. Normally, setting the action routines in the menu item definition (within the menu object lists) will do.

**[pulldown]**Together with the menu definition, the complete code to set up a pulldown menu is written out. In the code generated with the Uobl option, only the action routine pointer of a menu item must be set to point to this pulldown code (action routines within this pulldown menu ... as above).

**[hexa]**All values are written out in hex numbers.

**[decimal]**All values are written out in decimal numbers.

**[Jobname]**Only available if [job] is selected. The job name is derivated from the file name and can be edited.

**[From]**File name of the input file. Default directory is proposed. Endings will be added automatically.

**[To]**Output file name. The name s derivated trom the input tile name by adding ‘\_asm’. Can be edited.

**DO**On a ‘DO’ in the EASYSOURCE window, the source code will be generated. It successful, ‘OK’ will be reported.

**1.3 EASYSOURCE V3.01**EASYSOURCE hastwo new items, which can be set to three different stages each:

Set action routine pointers to 0 (EASYMENC)   
All pointers to action routines are set to zero.   
This is the option which should be set to use a menu definition with the   
EASYMENC library routines.   
*Please notice:   
If you do not alter the menu definition in any way, you don’t need   
EASYSOURCE, as the definitions generated from EASYMENU are written out with all action routine pointers set to zero!*

Let action routines point to one pointer   
All loose menu item action routines are set to point to the same longword pointer at the end of the menu definition, using the word pointer to longword pointer option of the window manager with bit 0 set.   
Also all sub-window menu item action routines are set to point to a longword pointer, but always only one pointer for each sub-window:   
pointer to loose menu item action routines:

dc.w rout+0\*+1

…

;pointer application sub-window menu objects action routines:

dc.w rout+X-\*+1

…

label at the end of the menu definition:   
rout

end

You must then make sure, that one longword pointer for the loose menu item action, follows directly after the rout label:

dc.l loose\_act-\*

You may then switch from there using the loose menu item number wwl\_item to the appropriate routines.   
Now if the appropriate option as described below is used, pointers to application sub-window setup, draw, hit and control routines may follow, and then the pointer to the application sub-window menu item action routine:

dc.l applX\_act-\*

You may then switch from there using the menu item number wwm\_item to the appropriate routines.   
*Please notice:   
There is an entry reserved for each application sub-window regardless whether there is a menu object list for this sub- window or not!*

Let action routines point to list of pointers   
All action routines are set to point to a Iongword pointer, one for each item, at the end of the menu definition, using the word pointer to longword pointer option of the window manager with bit 0 set:   
thw rout÷0-t÷1 ;pointer to action routine item 0   
dc.w routt4\*+1 ;pointer to action routine item 1   
All sub-window menu item action routines are set to point to one pointer for each sub-window, as described above.   
rout   
end   
You must then make sure, that one longword pointer for each loose menu item action, follows directly after the menu definition code:   
dc.l looseo\_act-t   
dc.l loosel act-t   
Set internal routine pointers to 0 (EASYMENGI   
All pointers to application sub-window setup routines are set to point to a simple jmp wm.smenu(a2)   
All pointers to application draw, hit and control routines are set to zero.   
This is the option which should be set to use a menu definition with the   
EASYMENC Library routines.   
*Please notice:   
If you do not alter the menu definition in any way, you don’t need   
EASYSOURCE, as the definitions generated from EASYMENU are written out   
in this form!*Let internal routines point to default routines   
The loose menu item action routine pointers of items with selection key 3 or 5 are set to point to the default *escape* or *winmove* routines written out with the disassembly.   
All application sub-window routines are set to point to the default routines written out with the disassembly. These routine match the standard usages of a sub- window menu. Do not use this option, if you intend to control the menu with the EASYMENC or EASYMENCasm library routines, as the default routines do not fill the result area.   
Let internal routines point to list of pointers   
All application sub-window routines are set to point to a list of Jongword pointers, 4 for each sub-window

**111.2 Pointer Interface Library Routines**These routines match the correspQnding trap#3 operations of the pointer interface. They can be called via the standard qdos library too, but here the parameters can simply be passed as normal function parameters and must not be prepared and retrieved via the inregs/outregs structures.   
The routines allow low level access to pointer I/O in a window. This is performed on the data structure of the extended channel definition block, as described in chapter 5.   
Please notice that all other levels (window manager, easymenu) are based on this level, i.e. the higher level routines perform the necessary calls to these trap#3 operations together with other calls to standard I/O trap#3 operations.   
2.0 List of pointer inerface routines   
Ordered by trap #3 number Page   
SeC IOP\_FLIM 172   
$6D IOP\_SVPW 191   
$OE IOP\_RSPW 185   
5SF IOP\_SLNK 187   
$70 lOP PINF 178   
$71 lOP IRPTR 180   
$72 IOP\_RPXL 183   
$73 lOP WBLB 194   
$74 IOP\_LBLB 173   
$75   
$76 IOPWSPT 197   
$77 lOP\_SPRY 188   
$78   
$79   
$7A IOP\_OUTL 175   
$73 IOP\_SPTR 190   
$7C lop\_Pick 177   
. $7D IOP\_SWDF 193   
$7E OPWSAV 196   
$7F IOP\_WRST 195

Ordered logically Page   
Utility   
$6C IOP\_FLIM find window limits 172   
$6F IOP\_SLNK set bytes in linkage block 187   
$70 IOP\_PINF pointer information 178   
$7C OP\_PICK pick job on top 177   
Define   
$7A IOP\_OUTL define outline 175   
$7D IOP\_SWDF define sub-window sprite list 193   
Pointer control   
$71 IOP\_RPTR read pointer 180   
$78 IOP\_SPTR set pointer position 190   
***Save/restore***$6D IOP\_SVPW save part window 191   
$6E IOP\_RSPW restore part window 185   
$7E OP WSAV save window 196   
$7F IOP\_WRST restore window 195   
Graphics   
$72 IOP\_PPXL read pixel colour 183   
$73 OP WBLB write blob 194   
$74 OP LBLB write line of blobs 173   
$76 IOP\_WSPT write sprite 197   
$77 OP\_SPRY spray pixels in blob 188   
2.1 How the functions work   
The parameters are moved from the stack into the registers and then the corresponding trap#3 routine is called   
As required for C, registers D2-D7 and A2-A6 are always preserved. On return DO contains the function return value, which is zero for OK or a negative standard ODOS error code.

2.2 Error codes   
All functions return 0 if the function was completed without an error or a negative number, which is a standard QDOS error code. The standard meanings together with the old QDOS and the SMS2 nomenclature are given below. The meaning must always be seen in the context of the functions purpose. Special meanings are described with the function.   
gg QDOS SMS2 meaning   
O function completed without error   
-1 err.nc err,nc operation not complete   
-2 err.nj err.ijob invalid job ID   
err.om err imem out of memory   
.4 err.or err.orng out of range   
-5 err.bo err.bff I buffer overflow   
-6 err.no err.ichn channel not open   
-7 err.nf err.fdnf file or device not found   
.8 err.ex err.tex file already exists   
-9 erriu err.fdiu file or device in use   
-10 err.ef err.eof end of file   
-11 err.df err.drfl drive full   
-12 err.bn err.inam invalid name -   
-13 err.te err.tms transmission error   
-14 err.ff err.fmtf format failed   
-15 err.bp err.ipar invalid parameter   
-16 err.fe err.mchk file error (medium check error)   
-17 err.xp orr.iexp invalid expression   
-18 err.ov err.ovfl arithmetic overflow   
19 emni err.nimp not implemented   
-20 err.ro err.rdo read only   
-21 err.bl err.isyn bad line (invalid syntax)   
-22 err. rwf read write failed   
For further information about the QDOS error handling we refer to the QDOS (SMS2) Reference Manual (available from Jochen Merz Software).

2.3 Tips and tricks for C beginners   
The parameter declarations are examples to make clear which, and which type of parameters the functions expect. There often are other equivalent declarations possible, e.g. a buffer area could be asubset of a larger area.   
We often expect the address of a parameter passed to the function, to allow the function to fill in or alter the parameter itself (which might be an address pointer). Where possible, normally the address off operator & is used for this purpose. Please notice, that as a C language speciality after:   
short buffer[1 0];   
*buffer* itself is a *variable* which contains the address pointer to the array and is equivalent to the address of the first element using the address off operator &:   
buffer == &buffer[0].   
Be aware: buffer-il does not increment the address by one, but points to the   
second element: buffer÷1 =&buffer[1].   
Please also remark that the numbering of array elements in C starts from zero, thus an array declared to have 10 elements, e.g. short buffer[10], numbers from 0 to 9!   
To avoid time and memory consuming passing and copying back of parameter values, we sometimes use the C language possibilty, to declare parameters or structures such that only space for an address pointer variable is reserved, which is then filled in by a library routine, e.g. if we define   
short (\*buffer)[1 0];   
then buffer is declared to be the address of the address pointer variable \*buffer which points to an array of ten short integers, and then &butfer is the address off this variable.   
Where such a declaration is found, C only reserves space for the variable and not the array. This allows us to fill in the adress in a function.   
Subsequently the calling C function may use the array quite normal. To make things a little easier for the programmer, a preprocessor instruction may help:   
#define buffer (\*bufferadr)   
short (\*bufferadr)[10]; /\*only reserves space for the pointer\*/ err=whereit.buffer(&bufferadr); /\* The function fills in the pointer. Now the array can be referred to by the name declared above, *e.g.\*I*buffer[0]=7;   
The round brackets in the above declaration are necessary, as   
short \*buffer[10];   
would only declare buffer to be an array of ten pointers of short integers.

2.4 Timeout   
All Pointer Interface functions allow a timeout to be passed. This is normally set to -1 = infinite, i.e. the function returns after the operation has completed. In some special cases a finite or even zero timeout can be set, e.g. in a RDPT call when only the pointer position is asked or if the function shall return after a certain time. A finite timeout is passed in units of 2Oms.   
If the timeout has elapsed before completing the operation, the function return value is-i (err.nc).   
2.5 QDOS channel ID   
The QDOS channel ID is passed as returned by the fgetchid() function.   
2.6 Primary window, secondary window, outline   
Before a read pointer call to a window channel can be made, the channels outline ***(->*** IOP\_OUTL) must be set. If the channel is a secondary window, also it’s primary outline must be set. Please read the chapters 2, 5 and 7 in part I of the manual.   
C programs normally have already some standard channels open. You may use the EU\_PWTST function of the EASYMENC Library to find an existing primary channel. All other channels already open or opend subsequently are secondary windows.   
2.7 Assembler   
The corresponding trap#3 operation call and return parameters are always given in short at the end of the function declaration.   
The assembler programmer thus may take advantage of this description and use the trap#3 calls directly.   
All registers not mentionned in the declaration are preserved.   
For further information we refer to the QPTR Technical Guide (available from Jochen Merz Software).   
2.8 Symbols used for call and return parameters   
*0* no return parameter   
= the parameter or register contents is unchanged   
7 the parameter or register contents is undefined   
**2.9 The routines**The following function definitions are in alphabetical order.

**lop FLIM**Description   
Returns the maximum possible outline size of the window. This is the size of the primary outline for a secondary window or the physical screen size for a primary window. The values returned in the *winlim* array are in absolute screen coordinates.   
Parameter declarations   
long err,chid   
short timeout,winlim[4];   
Function call   
err = iop\_Ilim (timeout,chid,winlim);   
Function return   
err error code ***(->*** below and 2.2)   
Parameters call return   
timeout -> 2.4 *0*chid channel ID *0*winlim address of result buffer *0*winlim [0] ? x size   
winlim [1] ? y size   
winlim [2] ? x origin   
winlim [3] 7 y origin   
Assembler: trap #3 iop.flim   
Register call return   
CO $6C error code   
EQ 0   
CO timeout   
AU channel ID   
Al pointer to result buffer   
Special error code meanings   
-15 err.ipar d2oO (assembler only)

**IOP\_LBLB**-> **IOP\_WBLB   
Description**Write a line of blobs to the given channel from the start pixel position to the end pixel position. The function searches along a chain of definitions until one matching the current screen mode is found, else a bad parameter error is returned. The function passes back the address of the actually used definition, therefore the pointer to the blob definition must be copied to a separate address pointer variable.   
Parameter declarations   
extern struct blob myblob; *1\** e.g. as generated with EASYSPRITE *I* extern struct pattern mypatt; *1\** e.g. as generated with EASYSPRITE *1*long err,chid,pmyblob;   
short xpos,ypos,timeout,xendp,yendp;   
Function call   
pmyblob=&myblob;   
err = iop\_lblb (xpos,ypos,timeout,chid,&pmyblob,&mypatt,&xendp,&yendp);   
Function return   
err error code ***(->*** below and 2.2)   
**Parameter call return**timeout -> 2.4 *0*chid channel ID *0*xpos origin x-pixel position 0   
ypos origin y-pixel position 0   
&xendp address of xendp variable 0   
xendp end x-pixel position end x-position   
&xendp address of xendp variable 0   
yendp end y-pixel position end y-position   
&pmyblob address of pmyblob variable 0   
pmyblob =&myblob address of blob used   
&myblob address of blob definiton *0*&mypatt address of pattern del intion *0*10 P\_LB LB

**.IOP\_LBLB   
Assembler trap #3 iop.lblb   
Register call return**lxi $74 error code   
Dl xly origin pixel position   
x1y end pixel position   
timeout   
PU channel ID   
Al pointer to (chain of) blob definitions pointer to blob def.   
*P2* pointer to (chain of) pattern detintions pointer to pattern def.   
Special error code meanings   
-15 err.ipar erroneous blob or pattern definition

EAS7PTR ***3* Library Routines Part III 175   
lop OUTL   
Description** -   
Define the outline of a window and declare the window to be managed by the pointer environment. All pointer relevant actions can only be done on a window if the outline is set ***(->*** EU\_UNMAN). The outline area defines the area within which pointer operations are possible. The outline area consists of hit area + shadow. A shadow can be drawn at any side of the window (in contrary to the shadow defined with a menu definition , which only allows a positive, right/bottom side shadow). The window area (set by iow.defw), which defines the area within which normal IC operations take effect, is restricted to be within the hit area of the outline. For secondary windows ICPOUTL also saves the window background (implicite IOP\_WSAV). The outline size and position can be changed with subsequent calls. If the size is kept unchanged, it is also possible to move a windows outline with it’s contents.   
The outline is restricted to be equal or smaller than   
- the screen size for a primary window,   
***-*** the primary windows outline for a secondary window ***(->*** ICP\_FLIM).   
Parameter declarations   
long err,chid;   
short xshadow,yshadow,keep,timeout;   
short outldef[4]={xsize,ysize,xorg,yorg}   
Function call   
err = iop\_outl (xshadow,yshadow,keep,timeout,chid,outldef);   
Function return   
err error code (-,- below and 2.2)   
Parameter call return   
timeout -> 2.4 ***0***chid channel ID ***0***xshadow x shadow size in pixels ***0***..—, yshadow y shadow size in pixels ***0***keep =1 keep window contents ***0***give up window contents ***0***outldef address of outldef array ***0***outldef[O] x size   
outldef[1] y size   
outldef[2] x origin   
outldef[3] y origin   
**...IOP\_OUTL**

.IOPOUTL   
Assembler trap #3 iop.outl   
Register call return   
$7A error code   
Dl x/y shadow width ?   
EP 1 keep contents, 0 give up contents   
timeout   
AD channel ID   
Al pointer to outline definition block   
Special error code meanings   
-4 err.orng area falls outside screen or outside primary outline

EASYPTR ***3*** Library Routines Part III 177   
**lOP\_PICK**Description   
The given jobs primary window is picked on top. Additionally a wake event can be sent to the window.   
Parameter declarations   
long err,jobid,chid;   
short wake,timeout;   
Function call   
err = op\_pick (jobid,wake,timeout,chid);   
Function return   
err error code ***(->*** below and 2.2)   
Parameter call return   
timeout -> 2.4 0   
chid channel ID 0   
jobid standard QDOS job ID 0   
or   
=—1 pick bottommost job   
=—2 mark unlockable   
wake =0 no wake 0   
=8 send wake event   
Assembler trap #3 iop.pick   
Register call return   
II $7C error code   
Dl job ID or key as defined above ?   
[P Oor8(sendwake) =   
[B timeout   
AD channel ID   
Special error code meanings   
-2 err.ijob invalid job ID or key

**IOP\_PINF**Description   
This functions gets information about the existance of the pointer interface and the window manager. Should always be called before any other pointer interface or window manager call is made.   
Returns the version number and the pointer to the window manager vectored routines. The window manager vectors ***(->*** Window Managar Library) are offsets from this address.   
Parameter declarations   
long err,chid,wrnan;   
short timeout;   
char version[4j;   
Function call   
err = iop\_pinf (timeout,chid,version,&wman);   
Function return   
err error code ***(->*** 2.2)   
Parameter call return   
timeout ->2.4 *0*chid channel ID 0   
version address of version variable 0   
version[O-3] ? version “n.nn”   
&wman address of wman variable 0   
wman base address of   
window manager   
Assembler trap #3 iop.pinf   
Register call return   
II $70 error code   
Dl ? version “n.nn”   
timeout   
ft13 channel ID   
Al ? base address of   
window manager   
Remark: If Al is zero after the call, then the window manager is not present!   
...IOP\_PINF

-Th

ASYPTR ***3*** Library Routines Part III 179   
.IOP\_PINF   
Special error code meanings   
-15 err.ipar no pointer interface present   
on return *from* the library routine only:   
-7 err.fdnf no window manager present

**lop RPTR   
Description   
Read pointer in** given window channel.   
Please note that a pointer read is only possible if the windows outline   
***(->*** IOP\_QUTL) has been set before. In the case of a secondary window, also the primary windows outline must be set.   
Also a pointer read on the primary window is only possible if there is no managed secondary window.   
On a secondary window a pointer read is only possible if it is the last opened and managed (outline set) secondary window.   
The IOP\_RPTR call returns if one of the events passed with the termination vector arises.   
The pointer position passed in xpos/ypos is used to determine whether the pointer was moved.   
Possible pointer events are:   
code meaning   
0 1 key or button stroke in window   
1 2 key or buffon down in window   
2 4 key or button up in window   
3 8 pointer moved from given xpos/ypos   
4 16 pointer out of window   
5 pointer in window   
6 64 reserved   
7 128 window request   
Multiple events are initiated by setting the appropriate bits (add code).   
Bit 7 is used to initiate a move window or change window size call. The move window sprite or the change window size sprite are shown at the actual pointer position instead of the actual window sprite. They can be moved over the whole screen. All other windows are locked. The call returns on a HIT or DO. It bit 7 is set, all other bits except bits 0 and 1 must be zero, and the call must be made with infinite timeout. Bits 0 and 1 then have the following meaning:   
bitmask code meaning   
10000000 128+0 show empty window sprite   
10000001 128+1 show move window sprite   
10000010 128+2 show change window size sprite   
10000011 128+3 show no sprite   
..IOP\_RPTR

..JOPRPTR   
Remark   
The pointer will return immediately:   
if both bits 4 and 5 are set (tvec =16+32=48)   
it the timeout is 0   
if xpos and/or ypos are outside of the screen   
This can be used to deteimine the actual pointer positionl   
The result of the pointer read is returned in the pointer record:   
actchid channel Id of the window the pointer actually is in   
subwin -l or the application sub-window number the pointer is in xposinwin x position of pointer in (sub-)window (relative window origin) yposinwin y position of pointer in (sub-)window (relative window origin) keycode =0, no key or button pressed, <>0 key (ASCII) or button code buttondown =0, no buffon down, <>0, button code   
sparev actually always zero   
winev window events (not used by IOP\_RPTR)(-> WM\_RPTR) subwinev sub-window events (not used by lOPRPTR)(-> WM\_RPTR) ptrev the pointer events which caused the return (bits or codes masked   
as defined above for tvec)   
windef (sub-)window size/origin   
Parameter declarations   
long err,chid;   
short xpos,ypos,tvec,timeout;   
struct prec {   
long actchid;   
short subwin;   
short xposinwin;   
short yposinwin;   
unsigned char keycode;   
unsigned char buttondown;   
unsigned char sparev;   
unsigned char winev;   
unsigned char subwinev;   
unsigned char ptrev;   
short windet[4];   
result;   
...IOP\_RPTR

**...IOP\_RPTR   
Function call   
err** = iop\_ rptr (tvec,timeout,chid,&result,&xpos,&ypos);   
**Function return**err ->2.2   
**Parameter call return**timeout ->2.4 *0*chid channel ID 0   
&result address of pointer record 0   
result pointer record   
Wee events to return 0   
&xpos address of xpos variable 0   
xpos old pointer x pos (absolute screen coord.) new absolute x 05   
&ypos address of ypos variable 0   
ypos old pointer y 05 (absolute ...) new absolute y pea   
Assembler trap #3 iop.rptr   
Register call return   
$71 error code   
Dl x/y pointer position (absolute screen ecord.) new position   
termination vector   
[B timeout   
*PD* channel ID   
Al pointer to pointer record

EASYPTR ***3* Library Routines Part III** 183   
**lop RPXL   
Description**Reads the pixel colour at a given position or scans until a certain colour is found.   
The coordinates are relative to the actual window area.   
Parameter declarations   
long err,chidscankey;   
short scancolour,tirneout,xpos,ypos,colour   
Function call   
err = iop.jpxl (scancolour,scankey,timeout,chid,&xpos,&ypos,&colour);   
Function return   
err ->a2   
Parameter call return   
timeout -> 2.4 *0*chid channel ID 0   
scancolour colour to find 0   
scankey see below 0   
&xpos address of xpos variable 0   
xpos start x-pixel position new x position   
&ypos address of ypos variable 0   
ypos start y-pixel position new y position   
&colour address of colour variable 0   
colour colour found   
scankey (assembler add 16 to the bit numbers)   
bit code / meaning if bit not set code / meaning if bit set   
15 0 no scan (read pixel colour) 32768 scan   
3 0 scan to different colour S scan until same colour   
2 0 scan horizontal 4 scan vertical   
1 0 scan up or left 2 scan right or down   
0 0 compare with start pixel colour 1 compare with scancolour   
...IOPRPXL

..JOP\_RPXL   
Assembler trap #3 iop.rpxl   
Register call return   
LB $72 error code   
Dl .1 Uy start pixel position new position/colour   
D2.l scan key/scan colour   
LB timeout   
channel ID   
The new position in the high word of Dl is the new x or y position according to the scan direction. The other dimension is unchanged. The scan key is in the most signiticant word of D2, so please add 16 to the bit numbers in the above scankey definition.

**lop RSPW**Description   
Restores part of a window contents from a save area as set up with **IOP\_SVPW.**The save area can be larger than the physical screen size.   
Parameter declarations   
long err,chid,savarea;   
short timeout;   
short savblock [4}{xstart,ystart,keep,keep};   
short winblock [4J={xsize,ysize,xorg,yorg};   
Function call   
err = iop rspw (timeout,chid,winblock,savarea,savblock);   
Function return   
err error code ***(->*** 2.2)   
Parameter call return   
timeout -> *0*chid channel ID *0*winblock address of window block parameters ***0***winblock[0] x size of block in window area =   
winblock[I] y size of block in window area =   
winblock[2] x origin position of block in window area =   
winblock[31 y origin position of block in window area =   
savarea pointer to save area =   
savblock address of save block parameters ***0***savblock[0j x start position of block in savarea =   
savblock[1j y start position of block in savarea =   
savblock[2] give up save area =   
<>0 keep save area =   
savblock[3] =0 give up save area =   
<>0 keep save area =   
**..IOP\_RSPW**

...IOP\_RSPW   
Assembler trap #3 iop.rspw   
Register call return   
ED seE error code   
Dl x/y start of block in save area   
<>0 keep save area, 0 give up save area   
lB timeout   
PU channel ID   
Al pointer to size/origin of window block   
P2 pointer to save area

**IOP\_SLNK   
Description** -   
**Set bytes in** linkage block. The pointer device linkage block is the data area of the pointer interface. Altering data in this area should not be performed by application programs, but there are two exceptions, these are the values which determine the mouse acceleration and wake up speed. The EU\_SMAWS function in the EASYMENC Library prepares the data and sets them using lOPS LN K.   
Parameter declarations   
long err,chid,linkblock;   
short timeout,linkpos,nurnber;   
char \*data;   
Function call   
*err* = op\_sInk (linkpos,number,timeout,chid,data,&linkblock);   
**Function return**en’ error code ***(->* 2.2)   
Parameter call return**timeout ->2.4 *0*chid channel ID *0*linkpos position in linkage block *0*number number of bytes to set *0*data address of data bytes *0*&linkblock address of Iinkblock variable *0*linkblock 7 linkage block address   
Assembler trap #3 iop.slnk   
Register call return   
$6F error code   
Dl.w position in linkage block   
•— D2.w number of bytes to set   
133 timeout   
PC channel ID   
Al pointer to data bytes linkage block address

**lop SPRY   
Description**Spray a number of pixels randomly in blob.   
There is no guarantee that the blob is filled entirely if the number of pixels to spray exceeds the number of pixels in the blob, or if the call is repeated.   
The routine searches along a chain of blob definitions, until a suitable definition for the actual screen mode is found.   
Parameter declarations   
extern struct blob myblob; /\* e.g. as generated with EASYSPRITE / extern struct pattern mypatt;   
long err,chid;   
short xpos,ypos,numpix,timeout;   
Function call   
err = op\_spry (xposypos,numpix,timeout,chid,&myblob,&mypatt);   
Function return   
err error code ***(->*** below and 2.2)   
Parameter call return   
timeout ->2.4 0   
chi’i channel ID ***0***XpOs origin x-pixel position ***0***ypos origin y-pixel position 0   
numpix number of pixels to spray 0   
&myblob address of blob definiton myblob 0   
&mypatt address of pattern defintion mypatt 0   
**fl**- **..IOP\_SPRY**

...IOP\_SPRY   
Assembler trap #3 iop.spry   
Register call return   
Lt $77 error code   
Dl Wy origin pixel position   
D2.w number of pixels to spray   
133 timeout   
ftfl channel ID   
Al pointer to (chain of) blob definition(s) pointer to blob def.   
pointer to (chain of) pattern defintion(s) pointer to pattern def.   
— Special error code meanings   
-4 err.orng Wy position is not in window, or blob falls outside   
-15 err.ipar bad data structure

**IOP\_SPTR**Description   
Set pointer position.   
Parameter declarations   
long err,chid;   
short xpos,ypos,keytimeout;   
Function call   
err = iop\_sptr (key,timeout,chid,&xpos,&ypos);   
Function return   
err error code ***(->*** below and 2.2)   
Parameter call return   
timeout -> 2.4 *0*chid channel ID ***0***&xpos address of xpos variable ***0***xpos x pixel position *(->* key) absolute x position   
&ypos address of ypos variable ***0***ypos y pixel position ***(->*** key) absolute y position   
key origin key   
=-1 coordinates are relative to current window definition   
=0 set to absolute screen coordinates   
=1 coordinates are relative to current hit area (-,.IOP\_OUTL)   
Assembler trap #3 iop.sptr   
Register call return   
ED $78 error code   
Dl x/y pixel position *(->* key) absolute position   
D2.b origin key (-1,0 or 1 as detined above)   
ED timeout   
PD channel ID   
Special error code meaning   
-4 err.orng position is not in window

**lOP SVPW**Description   
Saves part of a window contents. The save area can be larger than the physical screen size. The save area is setup or reused ***(->*** IOP\_RSPW).   
Parameter declarations   
long err,chid,savarea;   
short timeout;   
short savblock 4]={xstart,ystart,areaxsizeareaysize};   
short winblock [4]={xsizeysize,xorgyorg);   
Function call   
err = iop\_svpw (timeout,chid,winblock,&savarea,savblock);   
Function return   
error code ***(->*** below and 2.2)   
Parameter call return   
timeout -> 2.4 0   
chid channel ID 0   
winbiock address of window block parameters 0   
winblock(0] x size of block in window area =   
winblock[1] y size of block in window area =   
winblock[2] x origin position of block in window area =   
winblock[3] y origin position of block in window area =   
&savarea address of savarea variable 0   
savarea =0 set up new save area address of save area   
<>0 address of old save area =   
savblock address of save block parameters 0   
savblock[0] x start position of block in savarea =   
savblock[1] y start position of block in savarea   
savblock[2] reuse existing save area(savarea.cz>0)   
>0 x size of new savarea   
.—\_. savblock[3] reuse existing save area(savareaoo)   
>0 y size of new savarea   
..IOP\_SVPW

.-.loP—SvPw   
Assembler trap #3 iop.svpw   
Register call return   
ED $60 error code   
Dl xfy start of block in save area address of save area   
12 0 or xiy size of save area   
lB timeout   
AD channel ID   
Al pointer to size/origin of window block   
0 or address of save area(D2=0)   
Special error code meanings   
-4 er.orng block not in window or save area

rTh

**IOP\_SWDF**Description   
Set the pointer to the sub-window definition sprite list in the extended channel definition block. This list, a subset of the window working definition, defines the sub-window sizes and pointer sprites. For the main window (within the hit size) and each sub-window a separate sprite can be set.   
Recommendation   
Instead of setting up complicated structures, we recommend to define a menu   
definition using EASYMENU and draw it with the appropriate window manager or   
EASYMENC Library routines.   
Parameter declarations   
long err,chid;   
short timeout;   
struct splist mysplist;   
/ the structure definition can be found in chapter 5\*/   
Function call   
err = iop\_swdf (timeout,chid,&mysplist);   
Function return   
err ->2.2   
Parameter call return   
timeout ->2.4 *0*chid channel ID *0*&mysplist address of sub-window sprite list *0*Assembler trap #3 iop.swdf   
Register call return   
EU $7D error code   
EU timeout =   
channel ID   
pointer to sub-window sprite list

**IOP\_WBLB**Description   
Write a blob to the given channel. Searches along a chain of definitions, until one matching the current screen mode is found, else a bad parameter error is returned. -> IOP\_LBLB, IOP\_WSPT   
Parameter declarations   
extern struct blob myblob; / e.g. as generated with EASYSPRITE *J*extern struct pattern mypatt; / e.g. as generated with EASYSPRITE \*   
long err,chid   
short xpos,ypos,timeout;   
Function call   
err = iop\_wblb (xpos,ypos,timeout,chid,&myblob,&mypatt);   
Function return   
err error code ***(->*** below and 2.2)   
Parameter call return   
timeout ->2.4 *0*chid channel ID *0*xpos origin x-pixel position *0*ypos origin y-pixel position *0*&myblob address of blob definiton myblob *0*&mypatt address of pattern defintion mypatt 0   
Assembler trap #3 iop.wblb   
Register call return   
$73 error code   
Dl x/y origin pixel position   
0   
timeout   
PC channel ID   
Al pointer to (chain of) blob definition(s) pointer to blob def.   
1A2 pointer to (chain of) pattern defintion(s) pointer to pattern def.   
Special error code meanings   
4 err.orng Wy position is not in window, or blob falls outside   
-15 err.ipar bad data structure

**lop WRST**Description -   
This function restores the window contents previously saved with IOP\_WSAV or   
IOP\_OUTL.   
Parameter declarations   
long err,chid,savarea;   
short keep,tirneout;   
Function call   
err = iop\_wrst (keep,tinieout,ehid,savarea);   
Function return   
err ->12   
Parameter call return   
timeout ->2.4 *0*chid channel ID 0   
keep <>0 keep save area 0   
give up save area   
savarea pointer to save area 0   
restore from internal save area   
Assembler trap #3 iop.wrst   
Register call return   
LiD $7F error code   
D2.b <>0 keep save area, =0 give up   
[B timeout   
*P0* channel ID   
Al pointer to save area or 0 for internal area

**lop WSAV**Description   
Save the contents of the windows hit area.The save area can be supplied by the application or reserved internally. -> IOPWRST.   
Attention   
The internal save area may already be in use by the pointer interface to save the windows background for secondary windows *(->* IOP.\_OUTL) or the window contents for primary windows.   
Parameter declarations   
long err,chid,length,savarea;   
short timeout;   
Function call   
err = iop\_wsav (lengthtimeout,chid,savarea);   
Function return   
err error code -> 2.2   
Parameter call return   
timeout ->2.4 *0*chid channel ID *0*length length of given save area *0*take internal save area   
savarea pointer to save area *0*allocate or use internal save area   
Assembler trap #3 iopwsav   
Register call return   
$7E error code   
Dl 0 (Al <>0) or length of save area (Al =0) =   
timeout   
P0 channel ID   
Al pointer to save area (Dl>0)

**lop WSPT**Description   
Write a sprite to the given channel. Searches along a chain of definitions until one matching the current screen mode is found, else a bad parameter error is returned.   
For the 8 standard sprites of the pointer interface instead of the address a small number can be passed.   
Parameter declarations   
extern struct sprite mysprite;   
long err,chid;   
short xpos,ypos,timeout;   
Function call   
err = iop\_wspt (xpos,ypos,timeout,chid&mysprite);   
Function return   
err error code (> below and 2.2)   
Parameter call return   
timeout -> 2.4 *0*chid channel ID *0*xpos origin x-pixel position *0*ypos origin y-pixel position *0*&mysprite address of sprite definiton mysprite *0*or standard sprites   
o arrow   
1 padlock   
2 empty   
3 wrong mode   
4 keyboard entry   
5 no entry   
6 move window   
7 change window size   
...IOP\_WSPT

.IOPWSPT   
Assembler trap #3 iop.wspt   
Register call return   
ED $76 error code   
Dl x/y origin pixel position   
0   
ED timeout   
channel ID   
Al pointer to (chain of) sprite detinition(s) pointer to sprite def.   
Special error code meanings   
4 erromg Uy position is not in window, or sprite falls outside   
-15 err.ipar bad data structure

tTh