

III.4 The EASYMENU Control Library Routines

There are two libraries:

`easymenC_lib`

is intended for the C programmer as the routines in this library take the parameters from the stack. And

`easymenCasm_lib`

which is intended to be used from assembler, as these routines take the parameters from the registers.

Both, the assembler- and the C-programmer may take great advantage of the very easy way to create a menu controlled program.

The *easymenC_lib* routines follow the left to right parameter passing convention (see below) and thus can be used with the Metacomco Lattice C and the C68 compilers as well (and naturally with any other C-compiler using the same convention).

If these routines are called from assembler (which is also possible), the assembler programmer has to ensure himself that the parameters are passed on the stack in the right order.

The description of the functions is valid both for C and assembler.

Attention:

The routines have different prefixes. The meanings are

Prefix **EM**

All EM_ routines are standard routines to setup and control a menu definition.

These routines must only be used on a menu setup with EM_SETUP

Prefix **EU**

All EU_ routines are utility routines which are not directly coupled with a menu definition.

Prefix **EW**

All EW_ routines can be used on any standard menu working definition, i.e. on a definition set up with EM_SETUP or on a definition set up using the standard window manager routines as well.

Prefix **EI**

The library also contains these routines, which prepare parameters for internal routines written in C. The descriptions can be found in chapter 3.

4.0 List of EASYPTR Library routines

Ordered logically	Easy menu management	Page
Setup		
EM_SETUP	setup complete working definition	262
EM_CLEAR	remove EM_SETUP generated menu	257
Draw		
EM_IDRAW	draw info windows	258
EM_LDRAW	draw loose menu items	259
EM_MDRAW	draw application sub-window	260
EM_WDRAW	draw (redraw) complete menu window	272
Main menu read		
EM_RPTR	EM_SETUP generated menu read	261
Sub-window menu		
EM_AMCLR	remove sub-window menu	255
EM_AMSET	set up dynamic sub-window menu	256
Set WINDOW area		
EM_SWAMI	set window to sub-window menu item	264
EM_SWAPP	set window to application sub-window	265
EM_SWHIT	set window to hit area	266
EM_SWINF	set window to information sub-window	267
EM_SWIOB	set window to information object	268
EM_SWLIT	set window to loose menu item	269
EM_SWLNK	link other channel to actual window	270
EM_SWSEC	set window to sub-window menu section	271
General menu management utility routines		
EW_AWINF	get application sub-window address	281
EW_AWSCT	set menu section control blocks	282
EW_CLCHG	clear status bytes change bits	283
EW_DRBDR	draw current item border	284
Utility routines		
EU_CSTRQ	copy C string to QDOS string	273
EU_PWMAK	make primary window	274
EU_PWTST	find primary window	275
EU_QSTRC	copy QDOS string to C string	276
EU_RMAWS	read system mouse parameters	277
EU_SLEEP	use button_sleep thing (if QPAC2 present)	278
EU_SMAWS	set system mouse parameters	279
EU_UNMAN	set window channel to unmanaged stage	280

4.1 Internal routines (C only)

Principally the handling of internal routines by the EASYMENC Library Routines is the same as described in chapter 3.1.

Additionally, if a pointer to an internal routine in the EASYMENU generated menu definition is zero, then EM_SETUP will replace it by it's own routine. If the pointer is not zero, then EM_SETUP assumes that you have your own routine there and sets this routine.

More detailed information about the handling of action routines is given in 4.5.

4.2 Register A5 and A6

Registers A5 and A6 are in no way affected by the EASYMENU Library Routines. Thus e.g. they can be used to point to private data areas accessible from within the internal setup or action routines.

For C this principally means, that local parameters of the calling routine would be accessible from within internal routines.

4.3 Address Pointers (C only)

For QDOS we generally assume address pointers to be long integers!

4.4 EASYMENCasm Library routines (assembler only)

To make life a little bit easier for assembler programmers a special

easymenCasm_lib

library is available, which contains special variants of the EM_, EU_ and EW_ routines.

These take the parameters directly from the registers.

Routines not present here can be used either by pushing the parameters or use of the window manager routines directly (e.g. wm, swapp, ...)

There are two labels for each routine, one where the underscore is replaced by a full stop and one by an X. This is to allow usage of the routines with assemblers which do not allow a full stop in a label name.

The easymenCasm_lib library has external references into the easymenC_lib library.

Therefore, both libraries must be present in the linker control file:

LIBRARY file_easymenCasm_lib

LIBRARY file_easymenC_lib

in this order!

Both the *easymenC_lib* library routines and the *easymenCasm_lib* library routines use exactly the same extended working definition.

Therefore, it is possible to use both types of routines within the same program!

The result area is accessed using the offset definition of the extended working definition (normally as offsets from a4) -> **keys_addwdef**

The buffer area is only intended for temporary data and may be used by the application program as well.

The result area is used to pass values back.

But the three longwords from w_flag and the linked list pointer w_link must never be changed!

EASYMENC extended working definition offsets

add.wdef	equ	\$70	
w_link	equ	-add.wdef	;long linked list start
w_buffer	equ	-\$6c	;buffer
;free	equ	-\$2c	;3 bytes
w_olmen	equ	-\$29	;used differently
w_lastwi	equ	-\$28	;4 words last appl item x/y size/origin
w_result	equ	-\$20	
w_ptrpos	equ	-\$20	;long: x,y ptrpos
w_keycde	equ	-\$1c	;word: key
w_eventum	equ	-\$1a	;word: event number
w_wintyp	equ	-\$18	;word: window type -2 event, -1 loose, >=0
w_itemno	equ	-\$16	appl
w_movdis	equ	-\$14	;word: item number
w_colrow	equ	-\$10	;long: x,y move distance
w_flag	equ	-\$0c	;long: column/row
w_event	equ	-\$0a	;word \$AFFE
w_event	equ	-\$0a	;byte: events to return
w_implch	equ	-\$09	;byte: -1 implicit channel, 0 given channel
w_chadr	equ	-\$08	;long: channel address
w_wman	equ	-\$04	;long: pointer to window manager

4.5 Action Routines

All loose menu and application sub-window menu items may have an action routine which is processed implicitly by the window manager routine WM.RPTR which is called by EM_RPTR.

The EM_SETUP routine allows a very flexible handling of action routines. In general there are four different ways to install an action routine:

- a) If the action routine pointer in the menu definition is zero, as it is in a menu definition generated from EASYMENU, then EM_SETUP will set a universal action routine, which saves the item data to the result area and then causes EM_RPTR to return
- b) If the action routine pointer is zero and selection key of a loose item is one of these standard events, then EM_SETUP will set an internal action routine:

<u>code</u>	<u>event</u>	<u>action</u>
3	cancel	remove menu definition
5	move	move window
6	change size	show change size sprite, after HIT set change data in result area and return
7	sleep	call button sleep thing (if QPAC2 present)

- c) If the action routine pointer is zero and the selection key of a loose menu item is the **ASCII code** (not the window manager event code) **of the key** generating the events mentioned in b), then the selection key is changed to the standard event code and the universal action routine as in a) is set:

<u>ASCII code</u>	<u>event key</u>	<u>is changed to event code</u>
27	ESC	3
245	CTRL F4	5
241	CTRL F3	6
233	CTRL F1	7

Thus the items can be called with the above event keys but then a return from EM_RPTR follows. The program then may call an own action routine.

- d) If the action routine pointer is not zero, i.e. an action routine pointer has been set manually into the source listing generated with EASYSOURCE, then EM_SETUP will set this pointer.

4.6 Action routines written in C

As the window manager routine WM.RPTR, which processes the menu call, expects internal action routines to be written in assembler, it is only possible to use an action routine written in C if it is called from an assembler routine which prepares the parameters for it and ensures the return parameters.

The method how this is can be done from within C, is described in chapter 3.1 and with -> WM_RPTR.

This method is only recommended for advanced programmers.

The easier method is to call the action routines after EM_RPTR. Therefore, the data in the result area is used to decide which action routine is to be called.

4.7 Primary window (C only)

Normally your C-Program will have the standard I/O channels stdin, stdout and stderr already open. It depends on your compiler and eventually special parameters whether these channels are opened as one or more QDOS channels.

As your compiler may additionally offer options to redirect these channels from the normal console channel (and you may have a good reason to do so), we can not rely on that there being a known primary window available.

To ensure the existence of a primary window, the EU_PWTST function can be called at the beginning of the program. This function returns the QDOS channel ID of an existing primary window channel or reports 'err.fdnf', if there is none. If you want to know which of the standard channels is the primary window, you must compare the found channel ID with the channel ID of the standard channel(s) using fgetchid().

Example:

```
extern long eu_pwtst();
main()
{
    long          err,pwchid,fgetchid();
    err=eu_pwtst(&primchid); /*you should make an error test */
    if (err==-7)
        goto newpw; /* no primary window found ->open new*/
    if (err<0)      /*any other error should be handled by*/
        myerror(); /*your error routine*/
    /* if you want to know which of the standard channels is the primary */
    errchid=fgetchid(stderr);
    if (errchid==pwchid)
        goto foundpw; /* and so on */
newpw: ...
...
foundpw: ...
```

Another way to get control over the primary window, is to call the EU_PWMAK function once at the beginning of the program. This creates a new primary window and leaves all other window channels, which are already open, unchanged (as secondary windows).

Example:

```
extern long eu_pwmak();
main ()
{
    long err,pwchid;
    err=eu_pwmak(&pwchid); /*now pwchid is the primary window*/
    ...
```

4.8 The routines

The routines follow in alphabetical order.

EM_AMCLR

Description

Clear application sub-window menu.
All internal routine pointers as well as an existing menu structure are removed.
If the menu was set up with EM_AMSET, then also the memory reserved for row-
lists, status- and control-areas is released.

Parameter declarations

long err,subwdef,workdef;

Function call

err = em_amclr (subwdef,workdef)

Function return

err -> 2.2

Parameter	call	return
subwdef	address of sub-window definition	0
workdef	address of working definition	0
Assembler	em_amclr	emXamclr
d0	?	error code
a3	pointer to sub-window definition	=
a4	pointer to working definition	=

EM_AMSET

Description

Setup application sub-window menu.

This only works if there is no menu structure in the sub-window (->EM_AMCLR).

Use EWAWINF to find the address of the sub-window.

This routine should not be called from within the application sub-window setup routine or any other internal routine.

The routine sets default internal routine pointers, as described for EM_SETUP.

The spacing lists for each dimension consist of two short integer values for each column (row) giving the hit size and the spacing to the next element.

Please notice, that the spacing must at least be as large as the hit size + the current item border width (which must be doubled in x direction as usual).

Regular spacings can be given in one entry in negative values, e.g.

```
short      xspc[2]={-40,-42};
```

Please notice, that the bar and arrow colours are taken as defined with the window in EASYMENU, but they can be altered using the application sub-window structure definition.

If the pointer to the menu object list is zero, then only the pan/scroll bars are set up. Use **EM_MDRAW** to draw the menu.

Parameter declarations

```
short      nxsc,nysc,ncol,nrow;
short      xspc[ncol][2],yspc[nrow][2];
long       err,subwdef,workdef;
struct     wwm_mobj  mobjlist[nrow][ncol];
```

Function call

```
err: em__amset (nxsc,nysc,ncol,nrow,xspc,yspc,mobjlist,subwdef,workdef)
```

Function return

err -> 2.2

Parameter	call	return
nxsc	maximum number of x sections	0
nysc	maximum number of y sections	0
ncol	number of columns	0
nrow	number of rows	0
xspc	address of x (column) spacing list	0
yspc	address of y (row) spacing list	0
mobjlist	address of contiguous menu object list	0
subwdef	address of sub-window definition	0
workdef	address of working definition	0

EM_IDRAW

Description
Draw (redraw) one or several of the first 32 information sub-windows.
Each window is represented by the corresponding bit in the select parameter and is redrawn, if the bit is clear, e.g.
select=-2 /*\$FFFFFFFE, i.e. bit 2 clear*/
redraw window no. 2.

Parameter declarations
long err,select,workdef;

Function call
err = em_idraw (select,workdef)

Function return
err -> 2.2

Parameter	call	return
workdef	address of working definition	0
select	draw window(s) no. x if bit x clear	0

Assembler
Use wm.idraw.

Description
Draw (redraw) loose menu items.
If select is -1, then only those items are redrawn, which have the refresh bit (bit 0) in the status byte set.
All refresh bits are cleared.

Parameter declarations
short select;
long err,workdef;

Function call
err = em_ldraw (select,workdef)

Function return
err -> 2.2

Parameter	call	return
workdef	address of working definition	0
select	0 draw all	0
	-1 draw selected	0

Assembler
Use wm.ldraw.

EM_MDRAW

Description
Draw (redraw) an application sub-window menu.
The select flag decides whether all items or only those items which have the refresh bit (bit 0 in the menu item status byte) set are redrawn.

Parameter declarations
short select;
long err,subworkdef,workdef;

Function call
err = em_mdrawing(select,subworkdef,workdef)

Function return
err -> 2.2

Parameter	call	return
select	0 draw all	0
	-1 draw selected	0
subworkdef	address of sub-window definition	0
workdef	address of working definition	0

Assembler	em_mdrawing	emedrawing
d0	?	error code
d3	0 draw all, -1 draw selected	=
a0	?	chid
a3	pointer to sub-window definition	=
a4	pointer to working definition	=

EM_RPTR

Description

Start pointer read on a menu window setup with EM_SETUP. On return from EM_RPTR either an error is reported or if there was none, the result area can be used to decide what happened. The address of the result area was returned by EM_SETUP.

The return may be caused by an action or hit routine or a window event as passed with EM_RPTR in the event byte where each bit represents an event.

The window events passed with EM_RPTR overwrite internal window event action routines set by EM_SETUP, e.g. if there was an item with the selection key 5 in the menu definition, then EM_SETUP will have set the internal move window action routine for this item. But if with EM_RPTR the move bit in the event byte was set, then the internal move window action will not be performed and EM_RPTR will return.

Thus it is possible to decide with each call to EM_RPTR, whether an event shall be performed internally, cause a return (e.g. to perform a special action as required by the actual program status) or to be ignored.

Possible window events are:

Bit	Value	selection key	selection key code	meaning
0	1	DO/ENTER	2	DO
1	2	ESC	3	cancel
2	4	F1	4	help
3	8	CTRL F4	5	move window
4	16	CTRL F3	6	change size
5	32	CTRL F1	7	sleep
6	64	CTRL F2	8	wake

Parameter declarations

```
short      event;
long       err,workdef;
```

Function call

```
err = em_rptr (workdef,event)
```

Function return

```
err      -> 2.2      =27      menu removed
```

On return from EM_RPTR the result area is set -> 5.0.

Parameter	call	return
workdef	address of working definition	0
event	event to return (bit set as defined above)	0

...EM_RPTR

...EM_RPTR

Assembler	em.rptr	eeXrptr
d0	?	error code, 27 menu removed
a0	?	chid (? if d0<>0)
a1	?	pointer to status (? if d0<>0)
a2	?	pointer to wman (? if d0<>0)
a4	Pointer to working definition	=

EM_SETUP**Description**

Set up a working definition from an EASYMENU generated menu definition.

- use a given window channel or open a new one.
- calculate memory size of working definition and reserve memory
- setup menu structure from window definition
- if action-, draw-, hit- and control-routine pointers are zero, set default routines
- return size of actual layout and channel ID
- return address of status area, result area and working definition

The working definition is extended below its start for additional data. The complete format of this area is described in chapter 5.

Please notice that the structure of the working definition itself is absolutely standard, i.e. all standard window manager operations can be performed, but then the result area is not filled in.

All EM_ routines will report the error 'err.fdnf' if they are called with a working definition not set up with EM_SETUP!

...EM_SETUP**Parameter declarations**

```
short      xsize,ysize;
long       err,chid,status,result,workdef;
extern long menudef;
```

Function call

```
err: em_setup (&xsize,&ysize,&chid,&menudef,&status,&result,&workdef)
```

Function return

```
err      ->2.2
```

Parameter	call	return
&xsize	address of xsize variable	0
xsize	>0 x pixel size =0 take default size	layout x size
&ysize	address of ysize variable	0
ysize	>0 >y pixel size =0 take default size	layout y size
&chid	address of chid variable	0
chid	0 use or open primary window -1 open new window channel else use existing QDOS channel	QDOS channel ID
&status	address of status variable	0
status	?	address of status area
&result	address of result variable	0
result	?	address of result area
menudef	pointer to menu definition	0
&workdef	address of workdef variable	0
workdef	?	addr of working def
Assembler	em.setup	emXsetup
d0	?	error code
d1	x,y size or 0 or -1	x,y size
a0	chid or 0 or -1	chid
a1	?	pointer to status area
	a2 ?	pointer to window manager
a3	pointer to menu definition	=
a4	?	pointer to working definition

EM_SWAMI

Description
Set the WINDOW size and position of the menu channel to the last selected application sub—window menu item.

Parameter declarations
short applnum,stat;
long err,workdef;

Function call
err = em_swami (workdef,applnum,stat)

Function return
err -> 2.2

Parameter	call	return
workdef	address of working definition	0
applnum	application sub-window number	0
stat	item status which colours to set	0
	0 available colours	
	16 unavailable colours	
	128 selected colours	

Special error code meaning
-15 err.ipar last selection was not an application sub-window menu item

Assembler
Push parameters and call EM_SWAMI.

EM_SWAPP

Description

Set the WINDOW size and position of the menu channel to an application sub-window.

If ink== -1, then only the area is set, otherwise the paper and ink colours are set and the over status is set to 0.

Parameter declarations

short applnum,ink;
long err,workdef;

Function call

err = em_swapp (workdef,applnum,ink)

Function return

err -> 2.2

Parameter	call	return
workdef	address of working definition	0
applnum	application sub-window number	0
ink	-1 set window area only	0
	or	
	ink colour (the paper colour is taken from the window definition)	

Assembler

Push parameters and call EM_SWAPP, or use wm.swapp.

EM_SWHIT

Description

Set the WINDOW size and position of the menu channel to the actual hit area (the main menu area).

Parameter declarations

short paper,ink;
long err,workdef;

Function call

err = em_swhit (workdef,paper,ink)

Function return

err -> 2.2

Parameter	call	return
Workdef	address of working definition	0
paper	paper/strip colour to set	0
ink	ink colour to set	0

Assembler

Push parameters and call EM_SWHIT or use wm.swhit.

EM SWINF

Description

Set the WINDOW size and position of the menu channel to an information sub-window.

If ink== -1, then only the area is set, otherwise the paper and ink colours are set and the over status is set to 0.

Parameter declarations

short infonum,ink;
long err,workdef;

Function call

err = em_swinf (workdef,infonum,ink)

Function return

err -> 2.2

Parameter	call	return
workdef	address of working definition	0
infonum	information sub-window number	0
ink	-1 set window area only	0
	or	
	ink colour (the paper colour is taken from the window definition)	

Assembler

Push parameters and call EM_SWINF or use wm.swinf.

EM SWIOB

Description

Set the WINDOW size and position of the menu channel to an information object.

Parameter declarations

short	infunum,inobnum,ink;
long	err.workdef;

Function call

err = em_swio (workdef,infunum,ink,inobnum)

Function return

err -> 2.2

Parameter	call	return
workdef	address of working definition	0
infunum	information sub-window number	0
ink	ink colour to set	0
inobnum	information object number	0

Assembler

Push parameters and call EM_SWIOB.

EM_SWLIT

Description

Set the WINDOW size and position of the menu channel to a loose menu item. If stat== -1, then only the area is set, otherwise the paper and ink colours are set and the over status is set to 0.

Parameter declarations

short litmnum,stat;
long err,workdef;

Function call

err = em_swlit (workdef,litmnum,stat)

Function return

err ->2.2

Parameter	call	return
workdef	address of working definition	0
litmnum	loose menu item number	0
stat	-1 set window area only` or item status which colours to set 0 available colours 16 unavailable colours 128 selected colours	0

Assembler

Push parameters and call EM_SWLIT or use wm.swlit.

EM_SWLNK

Description

Set the WINDOW size and position of an unmanaged (outline not set) window channel other than the menu channel to the actual WINDOW size and position of the menu channel.

The WINDOW of the menu channel may have been set to the size and position of a menu element using EM_SWAMI, EM_SWAPP, EM_SWHIT, EM_SWINF, EM_SWIOB, EM_SWLIT or EM_SWSEC.

Parameter declarations

longerr,winchid,workdef;

Function call

err = em_swlnk (workdef,winchid)

Function return

err- > 2.2

Parameter	call	return
workdef	address of working definition	0
winchid	link channel ID	0

Special error code meaning

- | | | |
|----|----------|------------------------------------|
| -1 | err.nc | wrong working definition structure |
| -9 | err.fdiu | channel is managed. |

Assembler

Push parameters and call EM_SWLNK.

EM_SWSEC

Description

Set the WINDOW size and position of the menu channel to an application sub-window menu section
If ink== -1, then only the area is set, otherwise the paper and ink colours are set and the over status is set to 0.

Parameter declarations

short applnum,ink,xsecnum,ysecnum;
long err,workdef;

Function call

err= em_swsec (workdef,applnum,ink,xsecnum,ysecnum)

Function return

err -> 2.2

Parameter	call	return
workdef	address of working definition	0
applnum	application sub-window number	0
ink	-1 set window area only	0
	or	
	ink colour (the paper colour is taken from the window definition)	
xsecnum	x section number	0
ysecnum	y section number	0

Assembler

Push parameters and call EM_SWSEC or use wm.swsec.

EM_WDRAW

Description

Draw (redraw) a complete menu window structure
The initial position parameters are only relevant on the first call after EM_SETUP, i.e. they can be omitted for redraw.
If the position parameters are both given as -1, then the menu is drawn at the actual pointer position, i.e. the initial pointer position defined in the menu definition is positioned such that the systems pointer position must not be changed. If the menu window would fall outside the window limits (-> IOP_FLIM) the position is adjusted accordingly.

Parameter declarations

short xipos,yipos;
long err,workdef;

Function call

err = em_wdraw (xipos,yipos,workdef)

Function return

err ->2.2

Parameter	call	return
workdef	address of working definition	0
xipos	initial x pointer position	0
	-1 actual pointer position *	
yipos	initial y pointer position	0
	-1 actual pointer position *	

**both*

Assembler	wm.wdraw	wmXwdraw
d0	?	error code
d1	initial x,y position or -1	=
a4	pointer to working definition	=

EU_CSTRQ

Description

Copy C string (terminated by NUL) to QDOS string (with leading length word).

Parameter declarations

```
char      *cstring;
struct    qdosstring {
    short      length;
    char      string[];
    }         *qstring;
```

Function call

void eu_cstrq (cstring,qstring)

Function return

none

Parameter	call	return
cstring	address of C string	=
qstring	address of QDOS String	=

EU_PWMAK

Description

Try to make a known primary channel.
This only works if there is no or only still unused channels open, e.g. at the beginning of a C program after the startup module has only opened some standard channels.
Do not call EU_PWTST before EU_PWMAK, because then an existing channel might have been marked to be the primary.
The procedure will break if there is already an existing primary channel marked. In this case a new opened channel is closed again, and the error 'err.fdiu' is returned.

If the primary channel shall be accessible using standard C IO routines (e.g. fprintf,...), then a channel should be opened using fopen(), and the channel ID found with fgetchid() should be passed to EU_PWMAK.

Parameter declarations

longerr,pwchid;

Function call

err = eu_pwmak (&pwchid);

Function return

Err<0standard QDOS error code

Parameter	call	return
&primchid	address of primchid variable	0
primchid	-1 open new else take this	primary channel ID

Special error code meaning

-7	err.fdnf	no window manager present
-9	err.fdiu	another primary channel already exists
-15	err.ipar	no pointer interface present

Assembler	eu.pwmak	euXpwmak
d 0	?	error code
a 0	chid or-1	primary chid

EU_PWTST

Description

Searches the primary window of the job. If an existing primary channel is found, the outline is set to the actual size.

Parameter declarations

long err,pwchid;

Function call

err = eu_pwtst (&pwchid);

Function return

err <0 standard QDOS error code, e.g. err.fdnf if no channel found

Parameter	call	return
&pwchid	address of pwchid variable	0
pwchid	?	primary channel ID

Assembler	eu.pwtst	euXpwtst
d0	?	error code
a0	?	primary chid (? if d0<>0)

EU_QSTRC

Description

Copy QDOS string (with leading length word) to C string (terminated by NUL).

Parameter declarations

```
char          *cstring;
struct        qdosstring {
    short      length;
    char       string[];
}              *qstring;
```

Function call

void eu_qstrc (qstring,cstring)

Function return
none

Parameter	call	return
qstring	address of QDOS String	=
cstring	address of C string	=

EU_RMAWS

Description
Read system mouse acceleration and wakeup speed.
Should eventually be used before EU_SMAWS.

Parameter declarations
short accel,wakeup;
long chid;

Function call
err= eu_rmaws (chid,&accel,&wakeup)

Function return
err ->2.2

Parameter	call	return
chid	channel ID	0
&accel	address of accel variable	0
accel	?	actual mouse acceleration
&wakeup	address of wakeup variable	
wakeup	?	actual mouse wakeup speed

Assembler	eu.rmaws	euXrmaws
d0	?	error code
d1	?	actual mouse acceleration
d2	?	actual mouse wakeup speed
a0	chid	=

EU_SLEEP

Description

This function calls the button_sleep thing of QPAC2 if present.

Parameter declarations

long err;

Function call

err = eu_sleep ()

Function return

err -> 2.2

Parameter	call	return
none		

EU_SMAWS

Description

Set mouse acceleration and wakeup speed.
The function sets the corresponding values into the pointer interface linkage block. As this takes affect for the whole system, a program should eventually save the old values (-> EU_RMAWS) and restore them.

Parameter declarations

short accel,wakeup;
long chid;

Function call

err = eu_smaws (accel,wakeup,chid)

Function return

err ->2.2

Parameter	call	return
accel	mouse acceleration range 0 to 9	0
wakeup	mouse wakeup speed range 0 to 9	0
chid	channel ID	0

Assembler	eu.smaws	euXsmaws
d0	?	error code
d1	mouse acceleration	=
d2	mouse wakeup speed	=
a0	chid	=

EU_UNMAN

Description

Declare secondary channel to be unmanaged and give up internal save area if in use (->IOP_WRES).
This only works if there is no menu working definition installed (->WM_UNSET, EM_CLEAR).

Parameter declarations

long err,chid;

Function call

err = eu_unman (chid)

Function return

err -> 2.2

Parameter	call	return
chid	channel ID	0
Special error code meaning		
-1	err.nc	primary window
Assembler	eu.unman	euXunman
D0	?	error code
A0	chid	=

EW_AWINF

Description

Find address of application sub-window working definition and sub-window menu status area.
This routine is mainly intended to give access to the menu item status bytes, but also the sub-window structure in general.

Parameter declarations

long err,substatus,subworkdef,workdef
short winum;

Function call

err= ew_awinf (winum,&substatus,&subworkdef,workdef)

Function return

err -> 2.2

Parameter	call	return
winum	application sub window number	0
&substatus	address of substatus variable	0
substatus	?	menu sub-window status area
&subworkdef	address of subworkdef variable	0
subworkdef	?	address of sub-window working def.
workdef	address of working definition	0

Assembler	ew.awinf	ewXawinf
do	?	error code
d1	sub-window number	=
a1	?	pointer to menu status bytes
a3	?	pointer to sub-window def.
a4	pointer to working definition	=

EW_AWSCT

Description

This routine can be used to setup the application sub-window section control blocks. This routine should be called after the menu definition has been set up (e.g. with WM_SMENU). But it must be called with applwdef pointing to the same location (wwa_mstt) in the working definition as passed to EC_AWSET. Therefore, this pointer must be copied to applwdef before calling WM_SMENU, e.g:

```
long applwdef=subworkdef;
err=wm_smenu (xscale,yscale,status,wman,&subwindef,&subworkdef);
...
err=ew_awsct (startcol,startrow,panctrl,scrollctrl,applwdef,&subworkdef);
...
```

Parameter declarations

```
long            err,panctrl,scrollctrl,applwdef,subworkdef;
short           startcol,startrow;
```

Function call

```
err=ew_awsct (startcol,startrow,panctrl,scrollctrl,applwdef,&subworkdef);
```

Function return

```
err            0
```

Parameter	call	return
startcol	start column	0
startrow	start row	0
panctrl	pointer to pan Ctrl block	0
	>0 setup at this address	
	=0 setup in working definition	
scrollctrl	-1 use pointer (wwa_part) in sub-window working definition	
	pointer to scroll ctrl block	0
	>0 setup at this address	
	=0 setup in working definition	
	-1 use pointer (wwa_part) in sub-window working definition	
applwdef	pointer to application sub-window working menu definition as passed to EC_AWSET	0
&subworkdef	address of subworkdef variable	
subworkdef	running pointer to next free space in working definition	= or updated if panctrl or scrollctrl = 0

...EW AWSCT

Assembler	ew.awsct	ewXawsct
D0	?	error code
d1	start row	
d2	start column	
a1	pointer to pan control block or 0 or -1	pointer to pan ctrl block
a2	pointer to scroll control block or 0 or -1	pointer to scroll ctrl block
a3	ptr. to sub-window definition+\$64 (wwa_mstt)	=
a4	running pointer to working definition	updated (a3 = 0 only)

EW_CLCHG

Description

Utility routine to clear the change bits of the menu status bytes. These are not cleared by WM_MDRAW or WM_LDRAW!

Please notice that items which have the change bit set are not selectable, even if the status is available.

Parameter declarations

```
long err,subworkdef,workdef;
```

Function call

```
err = ew_clchg (subworkdef,workdef);
```

Function return

err -> 2.2

Parameter	call	return
subworkdef	pointer to sub-window definition >0 application sub-window =0 loose menu items	0
Workdef	pointer to working definition	0

Assembler	ew.clchg	ewXclchg
d0	?	error code
a1	?	pointer to item status bytes
a3	ptr. to sub-window definition, 0 loose	=
a4	pointer to working definition	=

EW_DRBDR

Description

This routine draws or clears the border around the current item. The routine uses or sets the current item information in the status area (and thus does a little bit more than the pure window manager routine).

Parameter declarations

long err,wman,workdef;
short colour;

Function call

err= ew_drbdrr (colour, wman,workdef);

Function return

err ->2.2

Parameter	call	return
colour	=-1 clear border	0
	0 to 255 draw border in this colour	
wman	address of window manager vector base	0
workdef	pointer to working definition	0

Assembler	ew.drbdrr	ewXdrbdrr
d0	?	error code
d1.l	border colour or -1 to clear	=
a0	?	chid
a1	?	pointer to status area
a2	pointer to window manager	=
a3	pointer to working definition	=

III.5 Data structures

5.0 Extended working definition, result area, item numbers

An additional work area is set up by EM_SETUP before the working definition. The format is defined as a structure definition in the header file easy_h.

This area also contains the result area which is the most important part. The address of the result area, which is defined as an array of short integers, is returned by EM_SETUP.

All item and sub-window numbers start from zero!

Result area

result[0]	x pointer position (absolute coordinates)
result[1]	y pointer position (absolute coordinates)
result[2]	key code
result[3]	event number (see below)
result[4]	return type
	-2 return caused by event
	-1 return from loose menu item action
	>=0 return from application sub-window hit, Ctrl, action sub-window number
result[5]	item number (numbering from zero)
	application sub-window menu items are numbered from top/left to bottom/right.
	on return from an action on the pan/scroll arrows or bars, a special pan scroll item number is returned (see below).
result[6]	x move distance after a move window/change size call.
	hit position on the bar, after an action on sub-window menu bars
	-1 hit on arrows if sub-window menu action
result[7]	y move distance after a move window/change size call
	length of the bar, after an action on sub-window menu bars
	-1 hit on arrows if sub-window menu action
result[8]	column number after sub-window menu item hit
	-1 hit on bars or arrows
result[9]	row number after sub-window menu item hit
	-1 hit on bars or arrows

Event numbers

8	sub-window split
9	sub-window join
10	sub-window pan
11	sub-window scroll
16	DO
17	cancel
18	help
19	move window
20	change window size
21	sleep
22	wake

Pan scroll item number

On return from the standard application sub-window menu control routine, the special pan scroll item number as generated by the window manager is returned in result[5]. Here all bits have a special meaning and should be masked out:

<u>bit(s)</u>	<u>value</u>	<u>Hex</u>	<u>meaning</u>
0-7			section number
8	256	\$0100	scroll down or pan right
9	512	\$0200	pan
10	1024	\$0400	DO on arrows or bar, or ALT SHIFT
up/down			
11	2048	\$0800	action on bar
12-15	28672	\$7000	always

Example:

result[5]==28929 hit on scroll down arrows in section 1

5.1 Menu window definition

```

/* This is the structure of a window manager menu definition.
 * As this definition contains short (word) pointers, it is not
 * suitable to be defined from within a C-module.
 *
 * EASYMENU generates window definitions in relocatable form, so
 * normally only the pointer to the definition must be defined
 * as an external label within the C-module which manages the menu.
 *
 * Anyway the description is given here in the form of
 * structure definitions.
 *
 * Assembler programmers may substitute
 *      long by          dc.l
 *      short by        dc.w
 *      char by         dc.b
 * to get the equivalent definitions in assembler.
 *
 * The definitions are mainly intended for informational or debugging
 * purposes.
 * Advanced programmers may adapt and use the definitions if a very complex
 * program requires this.
 *
 * Please notice that, if you want to replace the default
 * setup, draw, control or hit routines by C written routines,
 * you must initialize in_rout structures which simply consist
 * of an assembler instruction to jump to a suitable library
 * routine which in turn prepares parameters, calls the C-written
 * routine and retrieves parameters afterwards.
 *
 * Dimensional parameters must be altered accordingly
 */

#define UNLOCKED      0
#define NUMLAY        1

#if UNLOCKED
#define NUMIOB         0
#define NUMINFO        0
#define NUMLOOSE       0
#define NUMAPPL        0
#endif

```

```

/*****
#define      JSR          20153      /* = $4EB9 == jsr */
extern      long         ei_awsset();
extern      long         ei_awsdrw();
extern      long         ei_liact();
extern      long         ei_awswhit();
extern      long         ei_awsact();
extern      long         ei_awsctr();

struct      in_rout {
            short         jump;
            long          asm_rout;
            long          c_funct;
};
*/
/*      Example:
*      struct in_rout  aw1set = {JSR,ei_awsset,ec_awsset};
*/

/* Window Definition: */
#define      ENDMINUS      -1
#define      ENDNULL       0
#define      wda_clear      '\x0'
#define      wda_noclr      '\x80'

/* Structure to define window attributes */
struct      wd_wattr      {
            unsigned char  wda_clfg;      /*wda_clear or wda_noclr */
            char           wda_shdd;      /* shadow width */
            short          wda_borw;      /* border width */
            short          wda_borc;      /* border colour */
            short          wda_papr;      /* paper colour */
};

/* Structure to define the item attribute record */
struct      wd_atrec      {
            short          wda_back;      /*item background colour */
            short          wda_ink;        /*text ink colour*/
            short          wda_blob;      /*pointer to blob for pattern*/
            short          wda_patt;      /*pointer to pattern for blob*/
};

```

```
/* Structure to define the item attributes */
```

```
struct      wd_iattr      {
    short      wda_curw;    /* current item border width */
    short      wda_curc;    /* current item border colour */
    struct wd_atrec wda_unav; /* unavailable attributes */
    struct wd_atrec wda_aval; /* available attributes */
    struct wd_atrec wda_selc; /* selected attributes */
};
```

```
/* scaling flags*/
```

```
#define      scal_0_0      0          /* invariant, top nibble %0000 */
#define      scal_1_4      4096       /* scale 1:4, top nibble %0001 */
#define      scal_2_4      8192       /* scale 2:4, top nibble %0010 */
#define      scal_3_4      12288      /* scale 3:4, top nibble %0011 */
#define      scal_4_4      16384      /* scale 4:4, top nibble %0100 */
```

```
/* Repeated window definition record */
```

```
struct      wd_rept {
    short      wd_xmin;      /* layout x minimum size + scaling */
    short      wd_ymin;      /* layout y minimum size + scaling */
    short      wd_pinfo;     /* pointer to info sub-windows*/
    short      wd_plitm;     /* pointer to loose menu items */
    short      wd_pappl;     /* pointer to application sub-windows*/
};
```

```
/* Fixed part of window definition followed by NUMLAY layouts */
```

```
struct      windef      {
    short      wd_xsize;     /* window x size */
    short      wd_ysize;     /* window y size */
    short      wd_xorg;     /* window x origin */
    short      wd_yorg;     /* window y origin */
    struct wattr wd_wattr;   /* window attributes */
    short      wd_psprr;     /* pointer to pointer sprite */
    struct wd_iattr wd_lattr; /* loose menu item attributes */
    short      wd_help;      /* pointer to help window */
    struct wd_rept wd_rep[NUMLAY]; /* repeated layouts */
    short      endflag;      /*ENDMINUS */
};
```

```
/* Object types, TEXT may also be TEXT-charpos, where charpos is the
 * position of a character which shall be underlined.*/
```

```
#define      TEXT          0
#define      SPRITE        2
#define      BLOB          4
#define      PATTERN       6
```

```
/* Loose menu item record */
```

```
struct wd_litmr {
    short      wdl_xsiz;    /* hit area x size + scaling */
    short      wdl_ysiz;    /* hit area y size + scaling */
    short      wdl_xorg;    /* hit area x origin + scaling */
    short      wdl_yorg;    /* hit area y origin + scaling */
    char       wdl_xjst;    /* object x justification */
    char       wdl_yjst;    /* object y justification */
    char       wdl_type;    /* object type TEXT/SPRITE/... */
    unsigned char wdl_skey; /* selection key */
    short      wdl_pobj;    /* pointer to object */
    short      wdl_item;    /* item number */
    short      wdl_pact;    /* pointer to action routine */
};
```

```
/* The loose menu items list is just an array of NUMLOOSE loose menu items. *
 * the list is terminated by -1 */
```

```
#if UNLOCKED
```

```
struct wd_lili {
    struct wdjitm    wd_litms[NUMLOOSE];
    short            endflag;
    /*ENDMINUS */
};
#endif
```

```
/* Information object record */
```

```
struct wd_iobr {
    short wdo_xsiz;    /* object x size + scaling */
    short wdo_ysiz;    /* object y size + scaling */
    short wdo_xorg;    /* object x origin + scaling */
    short wdo_yorg;    /* object x origin + scaling */
    char  wdo_type;    /* object type TEXT/SPRITE/.. */
    char  spare0;      /* spare */
    short wdo_ink;     /* text ink colour or */
/*      short wdo_comb; pattern or blob to combine */
    char  wdo_csiz;    /* x character size */
    char  wdo_ycsz;    /* y character size */
    short wdo_pobj;    /* pointer to object */
};
```

/* The information object list is an array of information object records

* terminated by an end word of -1 */

#if UNLOCKED

```
struct wd_ioli {
    struct          w_iobr wd_iobs[NUMIOB];
    short           endflag;
};
#endif
```

/* Information window record */

```
struct wd_info {
    short wdi_xsiz; /* sub-window x size + scaling */
    short wdi_ysiz; /* sub-window y size + scaling */
    short wdi_xorg; /* sub-window x origin + scaling */
    short wdi_yorg; /* sub-window x origin + scaling */
    struct wattr wdi_watt; /* sub-window attributes */
    short wdLpobl; /* pointer to object list */
};
```

/* The information sub-window list is just an array of information

* sub-window records terminated by an endflag of -1 */

#if UNLOCKED

```
struct wd_iwli {
    struct          wdjnfo wd_infos[NUMINFO];
    short           endflag;
};
#endif
```

/* The application sub-window list, is a list of word pointers to

* application sub-window definitions */

#if UNLOCKED

```
struct wd_appli {
    short          wd_appls[NUMAPPL];
    short          ENDNULL;
};
#endif
```

```
/* Pannable or scrollable application sub-window control block definition.  
* The index definitions are not yet managed by the actual version of the  
* window manager!*/
```

```
struct wda_pctrl {  
    short          wda_part;          /* pointer to part window control  
                                      block for pan, scroll and split */  
    short          wda_insz;          /* index hit size + scaling */  
    short          wda_insp;          /* index spacing left of sub-  
                                      window + scaling */  
    short          wda_icur;          /* index current item border width */  
    short          wda_icurc;         /* index current item border colour */  
    struct wd_atrec wda_iiat;         /* index item attribute record */  
    short          wda_psac;          /* pan/scroll arrow colour */  
    short          wda_psbcb;         /* pan/scroll bar background colour */  
    short          wda_pssc;          /* pan/scroll bar section colour */  
};
```

```
/* Application sub-window definition */
```

```
struct wd_appl {
    short          wda_xsiz;      /* sub-window x size + scaling */
    short          wda_ysiz;      /* sub-window y size + scaling */
    short          wda_xorg;      /* sub-window x origin + scaling */
    short          wda_yorg;      /* sub-window y origin + scaling */
    struct wattr    wda_wattr;    /* sub-window attributes */
    short          wda_pspr;      /* pointer to pointer sprite */
    short          wda_setr;      /* pointer to setup routine */
    short          wda_draw;      /* pointer to draw routine */
    short          wda_hit;       /* pointer to hit routine */
    short          wda_ctrl;      /* pointer to control routine */
    short          wda_nxsc;      /* maximum number of x Ctrl
                                sections */
    short          wda_nysc;      /* maximum number of y Ctrl
                                sections */
    char           wda_skey;      /* sub-window selection key */
    char           wda_ext;       /* spare */
};
```

```
/* The control block for pannable sub-windows must only be present if
```

```
* wda_nxsc is not 0 */
    struct wda_pctrl pan_ctrl;
```

```
/* The control block for scrollable sub-windows must only be present if
```

```
* wda_nysc is not 0 */
    struct wda_pctrl srl_ctrl;
```

```
/* The menu definition block must only be present if there is a standard
```

```
* menu in the sub-window
```

```
*/
    short          wda_mstt;      /* pointer to menu status block */
    struct wd_iattr wda_iattr;    /* menu item attributes */
    short          wda_ncol;      /* number of actual columns */
    short          wda_nrow;      /* number of actual rows */
    short          wda_xoff;      /* x offset to start of menu */
    short          wda_yoff;      /* y offset to start of menu */
    short          wda_xspc;      /* pointer to x spacing list */
    short          wda_yspc;      /* pointer to y spacing list */
    short          wda_xind;      /* pointer to x index list */
    short          wda_yind;      /* pointer to y index list */
    short          wda_rowl;      /* pointer to menu row list */
};
```

/* The menu object spacing list is just an array of spacing records */

```
Struct      mospac      {
                short      wdm_size;      /* object hit size + scaling */
                short      wdm_spce;      /* object spacing + scaling */
};
```

/* The spacing list for each dimension is just an array of spacing records, e.g.

```
*
* struct      mospac aw1_xspc[NUMCOL] = {
*                {40,42},
*                {48,50},
*                ...
*                one for each column
*            };
```

* Regular spacings can be given in two negative short integers
* instead of a list, e.g.

```
* struct      mospac aw1_xspc ={-40,-42};
*/
```

/* The menu row list is an array of pointers to the row menu object lists start and end

```
*
* struct      mrowl      {
*                short      wdm_rows;      pointer to object row list start
*                short      wdm_rowc;      pointer to object row list end
*            }aw_mrowl[NUMROW];
*/
```

/* The menu object record is identical to the corresponding part
* in the loose menu item definition*/

```
struct      awmobj      {
char          wdm_xjst;      /* object x justification */
char          wdm_yjst;      /* object y justification */
char          wdm_type;      /* object type TEXT/SPRITE/... */
unsigned char wdm_skey;      /* selection key */
short         wdm_pobj;      /* pointer to object */
short         wdm_item;      /* item number */
short         wdm_pact;      /* pointer to action routine */
};
```



```
/* The NUMROW menu object lists for each row are an array of NUMCOL
 * menu objects. All objects can (but must not) be in one large list,
 * where the pointer to the row end points to the same address as the
 * pointer to the row start of the next row.
 */
```

```
#if UNLOCKED
struct wdm_objli {
    struct awmobj awmobjx[NUMCOL];
};
#endif
```


5.2 Window working definition

```

/* The structures of a window working definition are defined below.
 * As some of these require dimensional parameters which must be
 * defined according to the menu sizes, those definitions are locked! *
 * The window working definition is normally setup from a window
 * definition, but may also be setup directly using the structure
 * definitions defined below.
 *
 * EASYMENU generates window definitions which can be transformed
 * to a working definition by using the standard window manager
 * WM_SETUP routine or the EM_SETUP library routine.
 *
 * To allow access to the working definition, the structure can be
 * defined as a pointer, which is filled in by the setup routine.
 *
 * Some of the following definitions are determined by a dimension. To
 * allow usage for most applications, the dimensions are set so high
 * that they normally will not be exceeded.
 *
 ** Please alter according to your menu and unlock,
 * or define separately for each menu:
 */

```

```
#define UNLOCKED 0
```

```
#if UNLOCKED
```

```

#define NUMINFO 32 /* Maximum number of information windows */
#define NUMINOB 32 /* Maximum number of information objects */
#define NUMLITM 128 /* Maximum number of loose menu items */
#define NUMAPPL 32 /* Maximum number of application sub-windows */
#define NUMROWS 128 /* Maximum number of appl sub-window menu rows */
#define NUMCOLS 512 /* Maximum number of appl sub-window menu
                    columns */
#define NUMXSCT 8 /* Maximum number of appl sub-window x menu
sections */
#define NUMYSCT 8 /* Maximum number of appl sub-window y menu
sections */

```

```
#endif
```

```
/* Window attributes
```

```
* =====
*/
#define DOCLEAR    '\x0'        /* clear flag to clear window */
#define NOCLEAR    '\x80'

struct ww_attr {
    unsigned char  wwa_clfg; /* window clear flag */
    char           wwa_shdd; /* shadow width */
    short          wwa_borw; /* border width */
    short          wwa_borc; /* border colour */
    short          wwa_papr; /* paper colour */
};
```

```
/* Menu item attributes
```

```
* =====
*/

struct ww_atrec{
    short          wwa_back; /* item background colour */
    short          wwa_ink;  /* item ink colour */
    long           *wwa_blob; /* pointer to blob for pattern */
    long           *wwa_patt; /* pointer to pattern for blob */
};

struct ww_iattr{
    short          wwa_curw; /* current item border width */
    short          wwa_curc; /* current item border colour */
    struct ww_atrec wwa_unav; /* unavailable item attributes */
    struct ww_atrec wwa_aval; /* available item attributes */
    struct ww_atrec wwa_selc; /* selected item attributes */
};
```

```
/* Window working definition header and window definition
```

```

=====
*
* The definition normally is setup using wm.setup.
* The channel ID is filled in as passed to wm.setup.
* For a secondary window wm.pulld fills in ww_chid and sets
* ww_pulld to-1.
* The pointer position in ww_psave is used by wm.unset to restore
* the old position when the menu is removed.
* The pointer sd_wwdef in the extended channel definition block
* points to ww^splst. This pointer is filled in by wm.prpos or
* iop.swdf and cleared by wm.unset.
* ww_splst points to the application sub-window list.
*/

```

```

struct      wwd      {
/* header*/
    long      *ww_wstat; /* pointer to window status area */
    long      *ww_wdef; /* pointer to window definition */
    long      ww_chid; /* window QDOS channel ID */
    long      *ww_pprec; /* pointer to pointer record */
    long      ww_psave; /* saved pointer position */
    long      ww_spar1; /* spare */
    word      ww_spar2; /* spare */
    char      ww_spar3; /* spare */
    char      ww_pulld; /* pulldown flag */
    long      *ww_splst; /* pointer to sub-window sprite list */
/* window definition */
    short      ww_xsize; /* menu window x size */
    short      ww_ysize; /* menu window y size */
    short      ww_xorg; /* menu window x origin */
    short      ww_yorg; /* menu window y origin */
    struc      ww_attr   ww_wattr; /* menu window attributes */
    long      *ww_psptr; /* pointer to pointer sprite */
    struct      ww_iattr  ww_lattr; /* loose menu item attributes */
    long      *ww_help; /* pointer to help definition */
/* sub_window lists */
    short      ww_ninfo; /* number of information sub-windows */
    short      ww_ninob; /* number of information objects */
    long      *ww_pinfo; /* pointer to info sub-window list */
    short      ww_nlitm; /* number of loose menu items */
    long      *ww_plitm; /* pointer to loose menu items list */
    short      ww_nappl; /* number of application sub-windows */
    long      *ww_pappl; /* pointer to appl sub-windows list */

```

```
/* The main part of the working definition can (but must not) be followed
 * by the list to information sub-windows, information sub-window objects,
 * loose menu items, application sub-window list and application sub-window
 * definitions.
 *
```

```
/* Information sub-window definition
```

```
=====
*/
```

```
struct      wwi_info{

short          wwi_xsiz;    /* sub-window x size */
short          wwi_ysiz;    /* sub-window y size */
short          wwLxorg;     /* sub-window x origin */
short          wwi_yorg;    /* sub-window y origin */
struct ww attr wwi_watt;    /* sub-window attributes */
long           *wwi_pobl;    /* pointer to information object list */
};
```

```
/* The Information sub-window list is an array of information sub-window
 * definitions + 1 dummy element with wwi_xsize==-1 which signals
 * the end of the list
```

```
=====
*/
```

```
#if UNLOCK
```

```
struct wwi_info *wwi_infl[NUMINFO+1];
```

```
#endif
```

```
/* Information object definition
```

```

=====
* The object types for information objects and loose menu items are:
* -n    text underline n.th character
* 0      text no underline
* 2      sprite
* 4      blob
* 6      pattern
*/

```

```

struct wwo_inob{
    short      wwo_xsiz;    /* object x size */
    short      wwo_ysiz;    /* object y size */
    short      wwo_xorg;    /* object x origin */
    short      wwo_yorg;    /* object y origin */
    char       wwo_type;    /* object type */
    char       wwo_spar;    /* spare */
    union      wwo_txco{
        struct wwo_txta {
            short wwo_ink;    /* text object ink */
            char  wwo_csz[2]  /* text character x/y sizes */
        } wwo_txat;
        long *wwo_comb;    /* ptr to pattern or blob to combine */
    } wwo_txcv;
    long      wwo_pobj;    /* pointer to object /
};

```

```

/* The information object list is an array of information object
* definitions + 1 dummy element with wwo_xsize==-1 which signals
* the end of the list

```

```

=====
*/

```

```
#if UNLOCK
```

```
struct wwo_inob *wwo_iobl[NUMINOB+1];
```

```
#endif
```

```
/* Loose menu item definition
```

```

*=====
* The object types for information objects and loose menu items are:
* -n      text underline n.th character
* 0       text no underline
* 2       sprite
* 4       blob
* 6       pattern
*/

```

```

struct ww_litm{
short      ww_xsiz;    /* hit area x size */
short      ww_ysiz;    /* hit area y size */
short      ww_xorg;    /* hit area x origin */
short      ww_yorg;    /* hit area y origin */
short      ww_xjst;    /* object x justification rule */
short      ww_yjst;    /* object y justification rule */
char       ww_type;    /* object type */
char       ww_skey;    /* selection keystroke */
long       *wwl_pobj;  /* pointer to object */
short      ww_item;    /* item number */
long       *wwl_pact;  /* pointer to action routine */
};

```

```
/* The loose menu items list is an array of loose menu item definitions
```

```

* +1 dummy item where ww_l_xsize== -1 signals the end of the list
*=====
*/

```

```
#if UNLOCKED
```

```
struct ww_litm *wwl_litl[NUMLITM+1];
```

```
#endif
```



```

/* The application sub-window definition list is an array of pointers to application
 * sub-window definitions. The last element must be 0 to signal the end of the
 * list
 *=====*/
#if UNLOCK
long      *wwa_pappl[NUMAPPL+1];
#endif

/* Application sub-window definition */
*=====*/

struct      wwa_appl{
/* Main window definition */
    short      wwa_xsiz;      /* sub-window x size */
    short      wwa_ysiz;      /* sub-window y size */
    short      wwa_xorg;      /* sub-window x origin */
    short      wwa_yorg;      /* sub-window y origin */
    struct ww_attr wwa_watt;    /* sub-window attributes */
    long      *wwa_pspr;      /* pointer to pointer sprite */
    long      *wwa_draw;      /* pointer to draw routine */
    long      *wwa_hit;        /* pointer to hit routine */
    long      *wwa_ctrl;       /* pointer to control routine */
    short      wwa_nxsc;       /* maximum number of x sections */
    short      wwa_nysc;       /* maximum number of y sections */
    char      wwa_skey;        /* selection keystroke */
    char      wwa_spare1;      /* spare */
    short      wwa_spare2;     /* spare */
/* Pannable sub-window control definition */
    long      *wwp_part;       /* pointer to pan control block */
    short      wwp_insz;       /* index hit size */
    short      wwp_icur;       /* index current item border width */
    short      wwpjcic;        /* index current item border colour */
    struct ww_atrec wwp_iiat;   /* index item attribute record */
    short      wwp_psac;       /* pan arrow colour */
    short      wwp_psb;        /* pan bar background colour */
    short      wwp_pssc;       /* pan bar section colour */
/* Scrollable sub-window control definition */
    long      *wws_part;       /* pointer to scroll control block */
    short      wws_insz;       /* index hit size */
    short      wws_icur;       /* index current item border width */
    short      wws_icic;       /* index current item border colour */
    struct ww_atrec wws_iiat;   /* index item attribute record */
    short      wws_psac;       /* scroll arrow colour */
    short      wws_psb;        /* scroll bar background colour */
    short      wws_pssc;       /* scroll bar section colour */

```

```
/* Menu sub-subwindow definition */
```

```
    long      *wwa_mstt; /* pointer to menu status block */
    struct ww_iattr wwa_iatt; /* item attributes */
    short      wwa_ncol; /* number of columns */
    short      wwa_nrow; /* number of rows */
    short      wwa_xoff; /* x offset to start of menu */
    short      wwa_yoff; /* y offset to start of menu */
    union      wwa_xspa {
        short      wwa_xhsp[2]; /* -ve regular x hit size/x spacing */
        long      *wwa_xspc; /* pointer to x spacing list */
    }
    union      wwa_yspa {
        short      wwa_yhsp[2]; /* -ve regular y hit size/y spacing */
        long      *wwa_yspc; /* pointer to y spacing list */
    }
    long      *wwa_xind; /* pointer to x index list */
    long      *wwa_yind; /* pointer to y index list */
    long      *wwa_rowl; /* pointer to menu row list */
};
```

```
/* The menu object spacing list for each dimension is an array of short
 * integers for the object hit size and object spacing
```

```
=====*/
```

```
struct wwm_spac{
    short      wwm_size; /* object hit size */
    short      wwm_spce; /* object spacing */
};
```

```
/* A spacing list for N columns of a menu setup by the program may be
 * initialized as follows:
```

```

*
*      struct wwm_spac awl_xspl[NUMCOL]={
*          {col1size,col1spacing},
*          {...,...},
*          {colNsize.colNspacing}
*      };
*
* -----
```

```
/* on the other hand, if wwa_aw1 is a pointer to an existing structure
 * of type wwa_appl:
```

```
/* struct wwa_appl      wwa_aw1;
 * a pointer to an existing row spacing list can be defined as:
 * struct wwm_spac      *aw1_xspl[wwa_aw1->wwa_nrow];
 * and the pointer to the x spacing list is initialized as:
 * aw1_xspl=wwa_aw1->wwa_xspc;
 */
```

```
/* Menu object list entry definition
```

```

*=====
* The object types for menu items are:
* -n    text underline n.th character
* 0     text no underline
* 2     sprite
* 4     blob
* 6     pattern
*
* Items are numbered rowwise from top left to bottom right starting
* from 0.
*/

```

```

struct wwm_mobj{
    short    wwm_xjst;    /* object x justification rule */
    short    wwm_yjst;    /* object y justification rule */
    char     wwm_type;    /* object type */
    char     wwm_skey;    /* selection keystroke */
    long     *wwm_pobj;   /* pointer to object */
    short    wwm_item;    /* item number */
    long     *wwm_pact;   /* pointer to action routine */
};

```

```

/* The menu object list is an array of pointers to the start and end of the
* menu object list for each row.
* All menu objects can (but must not) be arranged in one list.
*=====
*/

```

```
#if UNLOCKED
```

```
struct    wwm_mobj    wwm_aw1r1[NUMCOL+1];
```

```
#endif
```

/* The menu row list is an array of pointers to the start and end of the menu
 * object list for each row

=====

* A menu row list entry is defined by:

*/

```
struct    wwm_mrow    {
    long        *wwm_rows;    /* pointer to row start */
    long        *wwm_rowe;    /* pointer to row end */
};
```

/* A menu row list for N rows set up by the program may be initialized
 * as follows:

```
struct wwm_mrow    aw1_mrow[NUMROW] = {
    {wwm_aw1 r1 ,wwm_aw1 r1+NUMCOL}
    {... ,.. },
    {wwm_aw1 rN,wwm_aw1 rN+NUMCOL}
};
```

* To access an existing definition, define pointers to the structures
 * and fill them in with the pointers found in the existing application
 * sub-window definition.

*/

/* For each pannable and/or scrollable sub-window, wwp_part and/or wws_part
 * point to a sub-window section control block:

=====

* For each section there is a sub-window section control block record

*/

```
struct    wss_schr {
    short        wss_spos;    /* section start pixel position */
    short        wss_sstt;    /* section start column or row */
    short        wss_ssiz;    /* section size (number of columns
                                or rows) */
};
```

/* The whole block for one dimension consists of a short integer which
 * contains the actual number of sections, followed by a record for
 * each section

```
struct    wss_part{
    short        wss_nsec;    number of actual sections
    struct    wss_schr    aw1_pcb[NUMXSEC];
};
```

*

* Please notice that there must be enough space for the maximum number of
 * sections (wwa_nxsc or wwa_nysc in the application sub-window definition). */

5.3 Window status area

```
/* The window status area is used to control an existing menu as defined
 * with the window working definition.
 * The status area is normally set up by wm.setup.
 * The control data is filled in or altered by wm.rptr and the action
 * routines in case.
 * The loose menu item status block, which is a byte for each loose menu
 * item, follows immediately after the window working area.
 * Other status or control areas, e.g. for application sub-window menus, may
 * also follow, but those pointers are held in the application sub-window
 * definition.
 */
```

```
#define NUMLITEM 512
```

```
/* The possible menu item statuses are defined in 'easy_h' */
```

```
/* The event vector is part of the pointer record as defined below.*/
```

```
/* Event vector pointer byte */
```

```
#define keyclick 1
#define keydown 2
#define keyup 4
#define ptrmove 8 /* pointer moved */
#define ptroutw 16 /* pointer out of window */
#define ptrinw 32 /* pointer in window */
```

```
/* Event vector sub-window byte */
```

```
#define split 1
#define join 2
#define pan 4
#define scroll 8
```

```
/* Event vector window byte */
```

```
#define DO 1
#define CANCEL 2
#define HELP 4
#define MOVE 8
#define CHGSZ 16 /* change size */
#define SLEEP 32
#define WAKE 64
```

/* The pointer record is part of the menu status area. The same
 * structure is used by the iop.rptr trap#3 routine

```

===== */
struct ptr_rec {
    long wsp_chid;    /* channel ID of window
                       the pointer is in */
    short wsp_swnr;   /* number of sub-window the
                       pointer is in or -1 */
    short wsp_xpos;   /* pointer x position relative
                       to main or sub-window origin */
    short wsp_ypos;   /* pointer y position relative
                       to main or sub-window origin */
    unsigned char wsp_kstr; /* code of key stroke or 0 */
    unsigned char wsp_kprs; /* code of key pressed or 0 */
    unsigned char wsp_evt;  /* event vector first byte 0 */
    unsigned char wsp_weve; /* event vector window byte */
    unsigned char wsp_seve; /* event vector sub-window byte */
    unsigned char wsp_peve; /* event vector pointer byte */
    short wsp_xsiz;    /* sub-window x size */
    short wsp_ysiz;    /* sub-window y size */
    short wsp_xorg;    /* sub-window x origin */
    short wsp_yorg;    /* sub-window y origin */
}
  
```

```
/* Menu status area
```

```
* =====*/
```

```
struct    ws_wstat {
    long          *ws_work;    /* pointer to working area */
    long          *ws_wdef;    /* pointer to window definition */
    struct ptr_rec ws_point;    /* pointer record */
    long          ws_ptpos;     /* absolute pointer position */
    short         ws_wmode;     /* display mode for this window */
    long          *ws_ciact;    /* pointer to current item action
                                routine */
    short         ws_citem;     /* current item sub-window number */
    short         ws_cibrw;     /* current item border width */
    short         ws_cipap;     /* paper colour behind current item
                                or border colour -> WM.DRBDR */
    short         wsp_cihxs;    /* hit area x size */
    short         wsp_cihys;    /* hit area y size */
    short         wsp_cihxo;    /* hit area x origin in sub-window */
    short         wsp_cihyo;    /* hit area y origin in sub-window */
    unsigned char ws_litem[NUMLOOSE]; /* loose menu item
                                status block */
};
```


5.4 Pointer Interface extended channel definition block

The pointer interface handles pointer operations in a window on an extended channel definition block. The size of this block is $30 = 48$ bytes. The block is inserted immediately after the channel definition header which is $18 = 24$ bytes long. The data should never be altered by application programs directly, but only using the appropriate pointer interface/I/O trap #3 routines.

```
struct    sd_extchd {
    short          sd_xhits;      /* x hit size */
    short          sd_yhits;      /* y hit size */
    short          sd_xhito;      /* x hit origin */
    short          sd_yhito;      /* y hit origin */
    short          sd_xouts;      /* x outline size */
    short          sd_youts;      /* y outline size */
    short          sd_xouto;      /* x outline origin */
    short          sd_youto;      /* y outline origin */
    union          primary {
        long       sd_prwlb;      /* primary linked list up */
        long       sd_pprwn;      /* pointer to primary window */
        long       sd_prwlt;      /* primary linked list down */
        long       sd_sewll;      /* secondary linked list */
        long       sd_wsave;      /* pointer to save area */
        long       sd_wssiz;      /* size of save area */
        long       sd_wwdef;      /* pointer to working definition */
        char       sd_wlstt;      /* window lock status */
        unsigned char sd_prwin;   /* bit 7 primary window
                                   bit 0 well behaved */
        char       sd_wmode;      /* window mode */
        char       sd_mysav;      /* -1 if implicit save area */
        char       sd_wmove;      /* window move flag */
    };
};
```

5.5 EASYMENC and EASYMENCasm extended working definition offsets

add.wdef	equ	\$70	
w_link	equ	-add.wdef	;long linked list start
w_buffer	equ	-\$6c	;buffer
;free	equ	-\$2c	;3 bytes
w_olmen	equ	-\$29	;used differently
w_lastwi	equ	-\$28	;4 words last appl item x/y size/origin
w_result	equ	-\$20	
w_ptrpos	equ	-\$20	;long: x,y ptrpos
w_keycde	equ	-\$1c	;word: key
w_evenum	equ	-\$1a	;word: event number
w_wintyp	equ	-\$18	;word: window type -2 event, -1 loose, >=0 appl
w_itemno	equ	-\$16	;word: item number
w_movdis	equ	-\$14	;long: x,y move distance
w_colrow	equ	-\$10	;long: column/row
w_flag	equ	-\$0c	;word \$AFFE
w_event	equ	-\$0a	;byte: events to return
w_implch	equ	-\$09	;byte: -1 implicit channel, 0 given channel
w_chadr	equ	-\$08	;long: channel address
w_wman	equ	-\$04	;long: pointer to window manager

5.6 EASYMENCasm assembler library routines overview

The *easymenCasm_lib* library contains special variants of the EM_, EU_ and EW_ routines. These take the parameters directly from the registers. Routines not present here can be used either by pushing the parameters or use of the window manager routines directly (e.g. wm.swapp ,...)

There are two labels for each routine, one where the underscore is replaced by a full stop and one by an X (for assemblers which do not allow a. in a label name).

The *easymenCasm_lib* library has external references into the *easymenC_lib* library. Therefore both libraries must be present in the linker control file:

LIBRARY f lpX_easymenCasm_lib

LIBRARY f lpX_easymenC_lib

in this order!

Both the *easymenClib* library routines and the *easymenCasm_lib* library routines use exactly the same extended working definition. Therefore, both type of routines can be used within the same program!

The result area is accessed using the offset definition of the extended working definition (normally as offsets from a4) ->**keys_addwdef**.

The buffer area is only intended for temporary data and may be used by the application program as well. The result area is used to pass values back.

The description of the functions is the same as for the *easymenC_lib* library.

Register d0 on return is always 0 or an error code if nothing else is mentioned. All registers not mentioned are preserved!

<u>Register</u>	<u>Call</u>	<u>Return</u>
d0	?	error code
em.amclr	emXamclr	
a3	pointer to sub-window definition	=
a4	pointer to working definition	=
em.amset	emXamset	
d3	number of columns	=
d4	number of rows	=
d5	maximum number of x sections	=
d6	maximum number of y sections	=
a0	pointer to x (column) spacing list	=
a1	pointer to y (row) spacing list	=
a2	pointer to menu object list	=
a3	pointer to sub-window definition	=
a4	pointer to working definition	=
em.clear	emXclear	
a4	pointer to working definition	=
em.mdraw	emXmdraw	
d3	0 draw all, -1 draw selected	=
a0	?	chid
a3	pointer to sub-window definition	=
a4	pointer to working definition	=
em.rptr	emXrptr	
d0		27 menu removed
d1	events to return	=
a0	?	chid (? if d0<>0)
a1	?	pointer to status (? if d0<>0)
a2	?	pointer to wman (? if d0<>0)
a4	pointer to working definition	=
em.setup	emXsetup	
d1	x,y size or 0 or -1	x,y size
a0	chid or 0 or -1	chid
a1	?	pointer to status area
a2	?	pointer to window manager
a3	pointer to menu definition	=
a4 ?	pointer to working definition	
em.wdraw	emXwdraw	
d1	initial x,y position or -1	=
a4	pointer to working definition	=

eu.pwmak	euXpwmak	
a0	chid or -1	primary chid
eu.pwtst	euXpwtst	
a0	?	primary chid
ew.awinf	ewXawinf	
d1	sub-window number	=
a1	?	pointer to menu status bytes
a3	?	pointer to sub-window def.
a4	pointer to working definition	=
ew.awsct	ewXawsct	
d1	start row	=
d2	start column	=
a1	pointer to pan control block (or 0,-1)	pointer to pan control block
a2	pointer to scroll ctrl block (or 0,-1)	pointer to scroll control block
a3	ptr. to sub-win. def.+\$64 (wwa_mstt)	=
a4	running pointer to working definition updated	
ew.clchg	ewXclchg	
a1	?	pointer to item status bytes
a3	ptr.to sub-window def. or 0 for loose	=
a4	pointer to working definition	=
ew.drbrdr	ewXdrbrdr	
d1.l	border colour or -1 to clear	=
a0	?	chid
a1	?	pointer to status area
a2	pointer to window manager	=
a4	pointer to working definition	=
eu.rmaws	euXrmaws	
d1	?	actual mouse acceleration
d2	?	actual mouse wakeup speed
a0	chid	=
eu.smaws	euXsmaws	
d1	mouse acceleration to set	=
d2	mouse wakeup speed to set	=
a0	chid	=
eu.unman	euXunman	
a0	chid	=