**111.3 The Window Manager Library Routines**These routines match the window manager vectored routines. They form a basic interface to manage menu structures. More comfortable routines to manage menu structures set up with EASYMENU can be found in the EASYMENC Library. There such things as calculating memory sizes, selling up complete structures, etc. are performed automatically.   
For informational purposes and for such rare cases where special requirements demand the direct use of the window manager vectored routines, we make them available as library routines.   
The data structures are described in chapter 5. Thus structures can be set up ty ***hand*** andlor manipulated as required.   
The base of the window manager vectors, which are simply offsets to entry points from there, can be found with the lop\_PINE function in the Pointer Interface Library (chapter 2.1).   
The window manager vectored routines can mainly be divided into setup, draw, access and utility routines.   
From within these routines, internal routines can be called via pointers in the menu definition.   
To make these internal routines accessible from within C, the El\_ library routines can be used. How this works is described with the routine definitions. The routines are in the separate EASYMENC Library.

3M List of window manager routines   
Ordered by vector number Page   
$04 WM\_SETUP copy window to working definition 229   
EI\_AWSET call internal application sub-window C setup 231   
$08 WM\_SMENU set up standard sub-window menu 233   
$OC W&PRPOS position primary window 218   
$10 WM\_PULLD open and position puildown window 220   
$14 WM\_UNSET unset working definition 241   
$18 WM\_WRSET set/reset working definition 246   
sic WM\_WDHAW draw menu window 243   
EI\_AWDRW call internal application sub-window C draw 244   
$20 WM\_MDRAW draw sub-window menu 213   
$24 WMINDEX draw sub-window menu bars and arrows 211   
$28 WMSWDEF set window to sub-window area 239   
$2C WM\_LDRAW draw loose menu item(s) 212   
$30 WM\_RPTR menu read pointer 222   
EI\_LIACT call internal loose menu item C action 223   
ELAWCTR call intenal application sub-window C control 227   
EI\_AWHIT call internal application sub-window C hit 225   
$34 WM\_MHIT standard sub-window menu hit routine 214   
FI\_AMACT call internal sub-window menu C action 215   
$38 WM\_PANSC standard sub-window menu control   
routine 219   
$3C WM\_IDRAW draw information sub-window(s) 210   
$40 WM\_CHWIN move window or get change size offset 205   
$44 WMDRBDP draw (undraw) border around current item 206   
$48 WtV1\_MSECT find standard sub-window menu section 217   
$4C WM\_STLOB set loose menu item object 235   
$50 WMSTIOB set information sub-window object 234   
$54 WM\_FSIZF find size of layout 209   
$58 WM\_SWINF set window to information sub-window 237   
$SC WM\_SWLIT set window to loose menu item 238   
$60 WM\_SWAPP set window to application sub-window 236   
$64 WM\_SWSEC set window to application sub-window section 240   
$68 WM\_RNAME read/edit name 221   
$6C WM\_ENAME edit name 207   
$70 WM\_UPBAR update pan/scroll bars 242   
$74 WM\_ERSTR get error code string 208

Ordered logically Page   
Setup   
$04 WM\_SETUP copy window to working detinition 229   
El\_AWSET call internal application sub-window C setup 231   
$08 WMSMENU set up standard sub-window menu 233   
$OC WM\_PRPOS position primary window 218   
$10 WM\_PULLD open and position pulldown window 220   
$14 WM\_UNSET unset working definition 241   
$18 WM\_WRSET set/reset working definition 246   
$54 WIvl\_FSIZE find size of layout 209   
Draw   
$1 C WM\_WDRAW draw menu window 243   
EI\_AWDRW call internal application sub-window C draw 244   
$20 WM\_MDRAW draw sub-window menu 213   
$24 WM\_INDEX draw sub-window menu bars and arrows 211   
$2C WMLD RAW draw loose menu item(s) 212   
$3C WM\_IDRAW draw information sub-window(s) 210   
$44 WM\_DRBDR draw (undraw) border around current item 206   
Set window area   
$28 WM\_SWDEF set window to sub-window area 239   
$58 WM\_SWINF set window to information sub-window - 237   
$sc WM\_SWLIT set window to loose menu item 238   
$60 WM\_SWAPP set window to application sub-window 236   
$64 WM\_SWSEC set window to application sub-window section 240   
Main menu read   
$30 WM\_RPTR menu read pointer 222   
El\_LIACT call internal loose menu item C action 223   
EI\_AWCTR call intenal application sub-window C control 227   
EI\_AWHIT call internal application sub-window C hit 225   
Sub-window menu control   
$34 WM\_MHIT standard sub-window menu hit routine 214   
EI\_AMACT call internal sub-window menu C action 215   
$38 WM\_PANSC standard sub-window menu control   
routine 219   
$48 WMJVISECT find standard sub-window menu section 217   
$70 WM\_UPBAR update pan/scroll bars 242   
Menu window change size or move   
$40 WM\_CH WIN move window or get change size offset 205   
Set object   
$4C WM\_STLCB set loose menu item object 235   
$50 WM\_STIOB set information sub-window object 234   
Utility   
$68 WM\_RNAME read/edit name 221   
$6C WM\_ENAME edit name 207   
$74 WMERSTR get error code string 208

3.1 Internal routines   
The window manager setup and control routines may call *internal routines* via pointers in the menu data definition which are expected to be supplied by the application program.   
These are the   
loose menu item action routines   
- application sub-window setup routines   
application sub-window draw routines   
- application sub-window hit routines   
application sub-window control routines   
and the   
application sub-window menu item action routines   
Managing these routines is a little bit difficult from within a C program, as the window manager passes the parameters to these routines in the registers, i.e. assumes that these routines are written in assembler. Thus writing internal routines in C, would require a small bit of assembler code, which moves the parameters from the registers to the stack, where they could be retrieved by the C-written routine.   
But there is an additional obstacle in managing these routines from within C, as the pointers to the internal routines in the menu window definitions are relative word pointers, while C can only generate absolute long word pointers resolved at runtime by the relocation module. This problem could be bypassed, it the C program would not use the menu window definition, but would set up the menu working definition at runtime directly, as here all pointers are absolute pointers. But setting up and managing a menu definition using complicated structure definitions from within a C program is not easy and some things, as the application sub-window setup, become even more complicated as from within an assembler written program.   
But it is possible to link a menu window definition written in assembler with a C program. Here even internal routines written in C can be linked using a structure definition which really is nothing else but a small bit of assembler code.   
Example:   
If you design a menu in EASYMENU and disassemble the definition using EASYSDURCE, then you have the assembler source code of this menu window definition. If there is an application sub-window in the definition, then there is already a standard application sub-window setup routine written out from EASYSOURCE, which might not be suitable for your programs purposes, e.g. if you want to set up a dynamic menu structure at runtime.   
Now you can write a setup routine in C. This routine must accept the parameters as defined for EC\_AWSET on page 232.

To connect this routine with the menu definition, a small bit of assembler code is necessary which calls a library routine to copy the parameters from the registers to the stack, then calls your C-routine and finally restores the registers and eventually copies back return parameters from your routine.   
This small bit of assembler code is supplied using a structure definition, which must be made outside any function:   
#define JSR 20153   
extem long ei\_awsetfl; P the library routine /   
long myCawsetQ; P your C written setup routine *\*1*struct in\_rout   
short jumpsub;   
long asm\_rout;   
long C\_rout;   
} awiset = fJSR,ei\_awset,myCawset);   
At runtime (after the C startup module has filled in the absolute addresses) the above defined awiset structure is nothing else but a jump to subroutine instruction to the library routine EI\_AWSET, followed by the absolute address of your C-written setup routine *myCawset.* The return address on the stack naturally is only used to get the address of the C routine.   
Now if you replace the section instruction in the menu definition source code by:   
section data   
and declare the above structure to be an external label   
xref awl set   
and then fill in the pointer to the application setup routine in the menu definition source code (and that is in fact all you have to do as an involuntary assembler programmer) as:   
dc.w awlset\* ;pointer to application sub-window setup   
and then assemble the definition, this pointer will be resolved as a reloctable word pointer correctly by the linker, but only if awlset is not more than 32768 bytes (which is the maximum for a signed number in a word) away. To make this quite sure, the menu definition should be placed in the same section as awiset, therefore we have replaced the section instruction above.   
In a large program even this might not be possible, but this problem can be bypassed with an additional bit of assembler code.   
The word pointers in the menu window definition can be deviated to long word pointers, as the window manager routines read a word pointer with bit 0 set (odd), as a pointer to a longword pointer after bit 0 has been cleared.

So, it you add a label and a long word pointer   
lawiset   
dc.l awl set\*   
at the end of the application sub-window definition and then fill in   
dc.w Iaw1set\*+1 ;pointerto application sub-window setup   
instead of the direct pointer to *awiset,* all should go well.   
To be on the safe side, you can do this generally for all pointers you fill in.   
The same method logically (i.e. with other names, structures and routines) works for all internal routines.   
IN\_ROUT structures can also be used to set or replace internal routines at runtime in the working definition. Then the structure must be defined static or outside any function and then be assigned to the appropriate stucture, e.g.   
static stnjct in\_rout \*awl hit;   
struct wwa\_appl \*N.walappl;   
wwal appl->wwa\_hit = awl hit;   
**3.2 The routines**The routine names are ordered alphabetically.

EASYPTR *3* Library Routines Part III 205   
**WM\_CH WIN**Description   
This routines checks the event vector in the status area. If the window move or change size event is set, the routine calls the IOP.RPTR trap#3 routine such that the window move or change size sprite is shown.   
After a hit, a window move operation is performed automatically, while for change size only the move distance is retumed. The change size operation itself must be done by the application, e.g. alter the menu size or set up a new menu.   
Parameter declarations   
long err,wman,workdef;   
short xdist,ydist,event   
Function call   
err = wm\_chwin (wman,workdef,&xdist,&ydist,&event);   
Function return   
err ->2.2   
Parameter call return   
wman address of window manager vector base *0*workdef pointer to working definition *0*&xdist address of xdist variable *0*xdist x dist distance   
&ydist address of ydist variable *0*ydist y dist distance   
&event address of event variable *0*event 0 or 20 (change size)   
Assembler wm.chwin   
Register call return   
ED 7 error code   
Dl ? x/y pointer move dist.   
D4.l ? 0 or 20 (change size)   
? channel ID   
A4 pointer to working definition

**WM DRBDR   
Description   
This routine draws or clears** the border around the current item. The routine uses the current item information in the window status area.   
To clear the current item, set ws\_citem, then if ws\_ciact is clear call wm.drbdr, else call ws\_ciact, clear ws\_ciact and call wm.drbdr.   
To set a current item, set the current item sub window number in ws\_citem (-I for a loose menu item or the application sub-window number), set the border width in wscibbrw, the border colour in ws\_cipap and the item hit area origin and size in wscihit. Then call wm.drbdr and reset ws\_cipap to the background colour. *Or use E W\_DRBDR instead.***Parameter declarations**long err,status,wman;   
Function call   
err = wmdrbdr (status,wman);   
Function return   
eir .> 2.2   
Parameter call return   
wman address of window manager vector base *0*status pointer to status area *0*Assembler $44 wm.drbdr   
Register call return   
cc ? error code   
PU channel ID   
Al pointer to status area

**WM ENAME**Description   
This routine can be used to edit a string in a given window channel. The string is written out from the actual cursor position in the window. The window width must be large enough to write out (edit) the string, i.e. the maximum number of editable characters is the number of characters which can be written out from the actual cursor position to the right side of the window minus 1 (for the termination character). The name buffer must also be at least of this size.   
The buffer is expected to contain a standard ODDS string, i.e. with a leading length word followed by the characters of the string.   
Please notice that C character strings do not follow this convention! WM\_ENAME puts the cursor at the end of the given string (->WM\_RNAME). Editing is performed with an intermediate buffer, i.e. the old string is only overwritten if the new string is accepted with ENTER or CURSDR key up or down.   
Editing can also be terminated by ESC, which leaves the old string unchanged. The termination character is returned as function return value.   
Parameter declarations   
long err,chid,wman;   
char \*buffer;   
Function call   
err = wm\_ename (chid,buffer,wman);   
Function return   
err 43 -> 2.2   
0 DK   
>0 termination character   
Parameter call return   
wman address of window manager vector base *0*chid channel ID 0   
buffer address of buffer 0   
buffer ODDS string new ODDS string   
Assembler sOC wm.ename   
Register call return   
ID error code or term .chr   
Dl.w ? termination character   
channel ID   
Al pointer to buffer

**WM\_ERSTR   
Description   
This routine finds** the QDOS error string to a given error code. This works with all AH, JM, JSJJSU, MG, Minerva and SMS2 versions of the QL operating system. The string is in the form of a standard QDOS string, i.e. a leading length word followed by the characters of the string.   
Please notice, that C stores strings in a different format!   
Therefore, if the address of the error string returned in *erstring* can not be used directly, i.e. by using a standard ODOS trap routine for output, the string must be copied to a C character string.   
**Parameter declarations**long err,wman,erstring   
short errcode;   
Function call   
err = wm\_erstr (errcode,wman,&erstring);   
Function return   
err ->2.2   
Parameter call return   
wman address of window manager vector base *0*errcode QDOS error code 0   
&erstring address of erstring variable 0   
erstring address of error string   
Assembler $74 wm.erstr   
Register call return   
**ED error code**Al pointer to error string

**WM\_FSIZE**Description   
This function is used to find the size of a layout in the menu window definition   
(see chapter 5 for the format).   
This can be used to find the corresponding memory size for the layout from a list.   
Parameter declarations   
long err,wman,windef;   
short xsize,ysize,laynum;   
Function call   
err = wm\_fsize (&xsize,&ysize,wman,windef,&laynum);   
Function return   
err ->2.2   
Parameter call return   
wman address of window manager vector base ***0***windef address of window definition ***0***&xsize address of xsize variable ***0***xsize x size in question layout x size \*)   
o -> take default x size   
&ysize address of ysize variable ***0***ysize y size in question layout y size \*)   
o -> take default y size   
&laynum address of laynum variable 0   
laynum layout number   
\*) the return layout size is   
= given size, if the size is marked to be scaleable   
e given size, if a suitable layout was found   
> given size, if it was smaller than the smallest layout   
Assembler $54 wm.f size   
Register call return   
DJ error code   
Dl x/y size actual xly size t)   
? layout number   
PS pointer to window definition

**WM IDRAW   
Description** -   
This routine draws the first 32 intormation sub-windows, e.g. after an information sub-window object has been changed ***(->*** WM\_STIOB)   
Window number x is redrawn, if the corresponding bit x is clear in the redraw parameter.   
**Parameter declarations**long err,wman,workdof;   
short redraw;   
Function call   
err =wm\_idraw (redraw,wman,workdef);   
Function return   
err -> 2.2   
Parameter call return   
wman address of window manager vector base *0*redraw bit x clear to redraw window no. x *0*workdet pointer to working detinition *0***Assembler $30** wm.idraw   
**Register call return   
7 error code**D3.l bit x clear to redraw window no. x   
**P0 channel ID**M pointer to working definition

**WMJNDEX   
Description**This routine draws the pan/scroll bars and arrows of a standard application sub- window menu.   
To redraw the bars after a pan/scroll operation, WM\_UPBAR can be called. *The routine originally was mainly intended to draw sub- window menu indices, but these are not yet implemented in the window manager.***Parameter declarations**long err, wman su bworkdef, workdef;   
Function call   
err = wm ndex (wman,subworkdef,workdef);   
Function return   
err ->2.2   
Parameter call return   
wman address of window manager vector base *0*subworkdef address of sub-window definition 0   
workdet address of working definition *0*Assembler $24 wm.index   
Register call return   
7 error code   
AD channel ID   
*P3* pointer to sub-window definition   
M pointer to working definition

**WM\_LDRAW**Description   
This routine draws all loose menu items. It select is -1, then only those items are redrawn, for which the change bit (bit 0) of the status byte is set. The routine does not clear the change bit, this must be done by the program itself before WM\_RPTR is called or before return from the loose menu item action routine. To do this, the EW\_CLCHG (->WM\_MDRAW) routine might be called.   
Please notice that an item which has the change bit set is not selectable, even if the status is available!   
Parameter declarations   
long err,wman,workdef;   
short select;   
Function call   
err = wm\_ldraw (select,wman,workdef);   
Function return   
err ->2.2   
Parameter call return   
wman address of window manager vector base *0*select 0 draw all items *0*-1 draw if status byte has change bit set   
workdef pointer to working definition *0*Assembler S2C wm.ldraw   
Register call return   
? error code   
Dab select flag (0 draw all, -Iselective) =   
PD channel ID   
A4 pointer to working definition

**WM\_MDRAW**Description   
This routine draws a standard sub-window menu. If select is -I, then only those items are redrawn, for which the change bit (bit 0) of the status byte is set. The routine does not clear the change bit, this must be done by the program itself before WM\_RPTR is called or before return from the application sub-window hit routine. To do this, the EW\_CLCHG routine might be called.   
Please notice that an item which has the change bit set is not selectable, even if the status is available!   
The routine is normally called the first time together with WMJNDEX from within the application sub-window draw routine ***(->*** WM\_WDRAW).   
Parameter declarations   
long err, wman , su bworkdef, workdef;   
short select;   
Function call   
err = wm\_mdraw (select,wman,subworkdef,workdef);   
Function return   
err ->2.2   
Parameter call return   
wman address of window manager vector base *0*select 0 draw all items *0*-1 draw if status byte has change bit set   
subworkdef address of sub-window definition *0*workdef address of working definition *0*Assembler $20 wm.mdraw   
Register call return   
? error code   
Rib select flag (0 draw all, -iselective)   
AD channel ID   
AS pointer to sub-window definition   
A4 pointer to working definition

**WM\_MHIT**Description   
The standard menu hit routine is normally called from within the application sub- window hit routine (->WM\_RPTR -> EI\_AWHIT) it a standard menu is present in the sub-window.   
The routine finds and marks the current item. If an item is hit, the action routine is called. If an action on the arrows has taken place, the application sub-window control routine is called *(->* WM RPTR -> EI\_AWCTR).   
The two dummy parameters in the parameter list must be present to bring WM\_MHIT to the same form as required for EC\_AWHIT (->WM\_RPTR).   
Parameter declarations   
long err,wman , su bwo rkdef ,workdef;   
short xpos,ypos,key,event,timeout;   
Function call   
err = wm\_mhit (xpos,ypos,key,event,O,O,wman,subworkdef,workdef,&tirneout);   
Function return   
err ->2.2   
Parameter call return   
wman address of window manager vector base *0*xpos pointer x pixel position (absolute coord.) *0*ypos pointer y pixel position (absolute coord.) *0*key Oor keystroke *0*event Oorl6(DO) *0*subworkdef address of sub-window working definition *0*workdet address of working definition *0*&timeout address of timeout variable *0*timeout ? -1   
Assembler $34 wm.mhit   
Register call return   
error code   
Dl x/y pointer position   
D2.w keystroke or 0   
23 -1   
D4.b 0 or 16 (DO)   
*P0* channel ID   
PS pointer to sub-window working definition   
pointer to working definition   
..WNLMHIT

NTERNAL ACIWON ROUTNES   
From WM\_MHIT the standard menu action routine is called, it an item was hit   
and the pointer to the action routine of this item is not 0.   
If the pan/scroll arrows where hit, the application sub-window control routine ***(->***WM\_RPTR -> ELAWCTR) is called.   
S SMACT   
**Description**As tor loose menu items, the item action routine may initiate the action itselt, just control the item status or cause a return from WM\_RPTR.   
Assembler Standsrd menu item action routine   
**Register** call return   
CCIR must be set   
*cxj* error code   
01.1 item column/row ?   
Daw item number   
D4.l 0 or 46 (DC) 0 or event to set   
channel ID =   
Al pointer to menu status block   
A2 pointer to window manager vector   
PS pointer to sub-window working definition ?   
P4 pointer to working definition =   
..WMMHIT

...WM\_MHIr   
ECSMACT   
Description   
If a standard menu action routine shall be written in C, the FI\_SMACT library   
routine can be used to call this routine using an IN\_ROUT structure   
(->WM\_RPTR -> C\_LIACT).   
Function definition   
ec\_smact (column,row,item,peventchid,menustat,wmansubworkdef,workdef);   
Parameter declarations   
long wman,chid,menustat,subworkdefworkdef;   
short column,row,item\*pevent;   
Function return   
error code   
Parameter call return   
wrnan address of window manager vector base *0*column column number of item hit *0*row row number of item hit *0*item item number (from top left to bottom right) 0   
pevent address of pevent variable 0   
\*pevent 0 or 16 (DC) 0 or event to set   
chid channel ID 0   
menustat address of menu status block 0   
subworkdef address of sub-window working definition 0   
workdet address of working definition 0

**WM\_MSECT**Description   
The routine is normally called from within the application sub-window hit routine   
(->WM\_RPTR ->El\_AWHIT) it a standard menu is present in the sub-window.   
The routine finds the current section and looks for pan/scroll arrow actions.   
Parameter declarations   
long err, wman , su bwo rkdef wo rkd et;   
short xpos,ypos,key,event,item,xsect,ysect;   
— Function call   
err = wm\_msect (xpos,ypos,key,&event,wman,subworkdef,workdef,&iteni, &xsect,&ysect);   
Function return   
err ->2.2   
Parameter call return   
wman address of window manager vector base *0*xpos pointer x pixel position (absolute coord.) *0*ypos pointer y pixel position (absolute coord.) *0*key 0 or keystroke *0*&event address of event variable *0*event for event number = or 10/pan, li/scroll   
subworkdef address of sub-window working definition *0*workdef address of working definition *0*&item address of item variable *0*item ? 0 or psin a   
&xsect address od xsect variable *0*xsect ? x section number a   
&ysect address od ysect variable *0*ysect 7 y section number   
Assembler $48 wm.msect   
Register call return   
error code, 0 or psin **1)**Di x/y pointer position (absolute)   
D2.w keystroke or 0   
123 7 x/y section number a   
D4.b event number of keystroke = or 10/pan, 11/scroll   
AS pointer to sub-window working definition   
P4 pointer to working definition   
° pan’scroll item number, as desrcibed for EI\_AWCTR (->WMRPTR).   
**2)** for an action on the pan/scroll arrows, the section number is returned as -i

**WM PANSC**Description -   
Pan, scroll, split and join operations of a standard application sub-window menu are done by this routine. WM\_PANSC is normally called from within the application sub-window hit or control routine (->WM\_RPTR ->El\_AWHIT).   
Parameter declarations   
long err,wman , su bwo rkdef ,wo rkdet;   
short item,barpos,barlen,event;   
Function call   
err = wm\_pansc (item,barpos,barlen,&event,wman,subworkdef,workdef);   
Function return   
err -> 2.2   
Parameter call return   
wman address ot window manager vector base *0*item pan/scroll item number *0*barpos position of hit on bar *0*barlen length ot the bar (pixels) ‘ ***0***&event address of event variable ***0***event pan or scroll event number 0   
subworkdef address of sub-window working definition ***0***workdef address of working definition *0*Assembler $38 wm.pansc   
Register call return   
? error code   
Daw item number’ ?   
03.1 position of hit or—i’ 7   
D4.b pan or scroll event’ D4.l = 0   
channel ID =   
AS pointer to sub-window working definition =   
A4 pointer to working definition =   
The pan/scroll item number, hit position and events are as desribed for   
EI\_AWCTR (->WMRPTR).

**WM\_PRPOS**Description \*   
Set (primary) window (outline) position, then call WM\_WRSET.   
It the origin is given as -1, the window is positionned such that the pointer must not be moved. The origin position in the window definition defines the initial pointer position within the window. If an origin is given, this is used instead of the actual pointer position to position the window in equivalent manner.   
Parameter declarations   
long err,chid,wman,workdef;   
short xorg,yorg   
Function call   
err =wrri\_prpos (xorg,yorg,&chid,wman,workdef);   
Function return   
eir ->2.2   
Parameter call return   
wman address of window manager vector base *0*xorg x origin or -i 0   
yorg yoriginor-i 0   
to position pointer relative, both   
xorg and yorg must be-i   
&chid address of chid variable *0*chid 7 QDOS channel ID   
workdef pointer to working definition 0   
Assembler SOC wm.prpos   
Register call return   
? error code   
Dl origin or-i   
P0 ? QDOSchannelID   
M pointer to working definition

**WM\_PU LLD   
Description**Open or use given secondary window, mark it ‘pull down” and then call   
WM\_PPPOS.   
If the origin is given as -1, the window is positionned such that the pointer must not be moved. The origin position in the window definition defines the initial pointer position within the window. If an origin is given, this is used instead of the actual pointer position to position the window in equivalent manner.   
In addition to the standard window manager routine, which always opens a new channel, the library routine also allows to reuse an existing window channel.   
Parameter declarations   
long err,chid,wman,workdef;   
short xorg,yorg;   
Function call   
err = wm\_pulld (xorg,yorg,&chid,wman,workdef);   
Function return   
err -> 2.2   
Parameter call return   
wman address of window manager vector base *0*xorg x origin or -1 *0*yorg y origin or -1 *0*to position pointer relative, both   
xorg and yorg must be -1   
&chid address of chid variable *0*chid channel ID (reuse) channel ID   
-1 open new   
workdef pointer to working definition *0*Assembler $10 wm.pulld   
Register call return   
[XI ? error code   
Dl origin or -1 =   
AD ? channel ID   
M pointer to working definition   
Attention: the window manager routine always opens a new channel!

**WM\_RNAME**Description   
This routine can be used to input/edit a string in a given window channel. The string is written out from the actual cursor position in the window. The window width must be large enough to write out (edit) the string, i.e. the maximum number of editable characters is the number of characters which can be written out from the actual cursor position to the right side of the window minus I (for the termination character). The name buffer must also be at least of this size. The buffer is expected to contain a standard QDOS string, i.e. with a leading length word followed by the characters of the string. Naturally this can be an empty string too.   
Please notice that C character strings do not follow this convention! WM\_RNAME puts the cursor at the start of the given string. If the first character is printable, the old string is stripped off and a new string can be entered. If the first character is SPACE, this is treated as ENTER!   
Editing is performed with an intermediate buffer, i.e. the old string is only over - written if the new string is accepted with ENTER or CURSOR key up or down. Editing can also be terminated by ESC, which leaves the old string unchanged.   
Parameter declarations   
long err,chid,buffer,wman;   
Function call   
err = wm\_rname (chid,bufter,wman);   
Function return   
err 4) ->22   
>0 termination character   
Parameter call return   
wman address of window manager vector base 0   
chid channel ID 0   
buffer address of buffer 0   
\*buffer QDOS string new QDOS string   
Assembler $68 wm.rname   
Register call return   
133 7 error code   
Dl.w 7 termination character   
channel ID   
Al pointer to buffer

**WM RPTR**Description   
Read the pointer within a menu window. The routine automatically marks the current item. The routine either   
- returns on a window event   
(if there is no appropriate loose menu item for this event),   
- calls the loose menu item action routine if such an item is hit,   
- calls the application sub-window hit routine   
(any time the pointer is moved into such a sub-window)   
o- calls the application sub-window control routine   
(if a standard menu with bars and/or arrows is present in the sub-window).   
The action or hit routines may initiate a return from wm\_rptr or perform the action coupled with the loose item or application sub-window directly.   
In the case of an application sub-window menu there are several window   
manager routines which may be called to assist managing the menu from within the hit or control routine.   
Parameter declarations   
long err,wman,workdef;   
Function call   
err = wmjptr (wman,workdef);   
Function return   
en’ -> 2.2   
Parameter call return   
wmari address of window manager vector base *0*workdef address of window working definition *0*Assembler $30 wm.rptr   
Register call return   
error code   
M pointer to working definition   
.WM\_RPTR

INTERAL ACTON ROUTNES   
In the menu definition pointers to internal action routines can be present. These routines are called from within WM\_RPTR.   
B UACT   
Assembler loose meru 1tem sIc4on FQLODC   
Register call return   
CCIR must be set   
133 ? error code   
Dl .1 Wy pointer position (absolute coordinates) ?   
D2.w upper cased keystroke   
D4.b event number 0 or window event   
PD channel ID   
Al pointer to status area ?   
P2 window manager vector ?   
P3 pointer to loose menu item ?   
*M* pointer to working definition   
Description   
The loose menu item action routine is called if a loose menu item is hit, either by a HIT, a DO or with it’s selection keystroke. The action routine may initiate the action, control the menu item status, and/or cause a return from WM\_RPTR. Returns are controlled via the error code and the event number.   
If on call the event code in D4 is not 0, then the item was hit by a window event keystroke. The routine may do the appropriate action or pass the code back to WMRPTR, which sets the corresponding event bit in the event vector ***(->*** pointer record -> status area).   
On the other hand the action routine itself can initiate an event, by seffing and passing back an event code in D4.   
If DO is not 0, WM\_RPTR also returns. Positive values can be used as flags, to switch to the appropriate action after WM\_RPTR.   
If there is no error and the event code in D4 is zero, then WM RPTR continues, i.e. an action routine which has done the job itself (don’t forget to set/reset the menu item status), must always end with   
moveq #O,d4 ;no event   
moveq #0,dO ;no error   
rts condition codes must be set   
..WM\_RPTR

..WM\_RPTR   
ECLÜACT   
Description   
It a loose menu item action routine shall be written in C, the El\_LIACT library routine can be used to call this routine using an IN\_ROUT structure, e.g.   
struct in\_rout Ii5act = {JSR,ei\_liact,ic5liact}   
where ic5liact is a routine as defined below:   
Function definition   
ec\_liact (xpos,ypos,key,pevent,chid,status,wman,itenidet,workdet);   
Parameter declarations   
long wman , chid status, item, workdet;   
short xpos,ypos, key, \*pevent;   
Function return   
‘0 error code   
o OK (continue WM\_RPTR)   
>0 flag (cause WM\_RPTR to return)   
Parameter call return   
wman address of window manager vector base 0   
xpos x absolute pointer position 0   
ypos y absolute pointer position 0   
key keystroke 0   
pevent address of event variable 0   
\*pevent event number of keystroke 0 or window event   
chid channel ID 0   
status address of status area 0   
itemdef address of loose menu item definition 0   
workdef address of working definition 0   
.WM\_RPTR

..WM\_RPTR   
ELAWfT   
Assembler app cation bindcw hit routine   
Register call return   
CCR must be set   
ED ? error code   
01.1 x/y pointer position (absolute coordinates) x/y pointer position   
D2.w upper cased keystroke   
=-1 hit by an external keystroke   
LB timeout   
D4.b event number (0, 16 or 17 only)   
*PD* channel ID   
Al pointer to status area   
*f2* window manager vector ?   
*PS* pointer to sub-window definition   
A4 pointer to working definition   
Description   
The application sub-window hit routine is called anytime the pointer is moved in or into an application sub-window or if the window is hit by a keystroke. The hit routine may initiate an action as required by the program or control a standard menu.   
If there is a standard menu in the sub-window, the hit routine may just call WM\_MHIT, which finds the current item and calls the standard menu action routine if an item was hit.   
If a menu can have several sections, WM\_MSECT can be called to find the current section and/or actions on the arrows.   
If an event shall cause WM\_RPTR to return, the appropriate bit in the event vector must be set.   
The timeout is normally set to —l (infinite), but it a program requires to detect a permanent button or key down (e.g. in a draw program), the timeout can be set to a finite value or even zero.   
...WM\_RPTR

...WM\_RPTR   
ECAWT   
Description   
it an application sub-window hit routine shall be written in C, the EI\_AWHIT library routine can be used to call this routine using an IN\_ROUT structure,eg.   
struct in\_rout awl hit = {JSR,ei\_awhit,iclawhit};   
where id awhit is a routine as defined below:   
Function definition   
ecawhit(xpos,ypos,key,pevenchid,statuswman,subworkdet,workdet,ptimeout); Parameter declarations   
long wman ciii status, subwo rkdet, wo rkdef;   
short xpos,ypos,key,\*ptimeout,pevent   
Function return   
error code   
Parameter call return   
wman address of window manager vector base 0   
xpos x absolute pointer position 0   
ypos y absolute pointer position 0   
key keystroke 0   
ptimeout address of timeout variable 0   
\*ptimeout ? timeout for next call   
pevent address of pevent variable 0   
\*pevent event number of keystroke 7   
chid channel ID ***0***status pointer to status area 0   
subworkdef pointer to sub-window working definition 0   
workdet pointer to working definition 0   
..WMRPTR

ELAWCTR   
Assembler application wb-window control routine   
Register call return   
CCR must be set   
II error code   
D2.w item number ?   
D3.l position ot hit or—i ?   
04.b pan or scroll event 0 or event to set   
AD channel ID   
Al pointer to status area   
window manager vector ?   
AS pointer to sub-window definition ?   
A4 pointer to working definition   
Description   
The application sub-window control routine is called from WMRPTR when the pan/scroll bars of a standard menu are hit or from WM\_MHIT when an action on the pan/scroll arrows took place.   
The routine is intended to control pan/scroll actions, i.e. call WM\_UPBAR or WM\_PANSC in the case of a standard menu.   
1) pan/scroll item number   
bit meanin   
0-7 section number   
8 scroll down/pan right   
9 pan   
10 DC on arrows (ALT SHIFT ***U),*** page pan/scroll   
ii hitonbars   
12 always set   
is always set   
14 always set   
15 always zero   
position of hit   
MSW hit position on bar   
LSW length of the bar (pixels)   
pan or scroll event   
10 pan   
ii scroll   
.WMRPTR

.WM\_RPTR   
Description   
If an application sub-window control routine shall be written in C, the EI\_AWCTR library routine can be used to call this routine using an IN ROUT structure,e.g.   
struct n\_rout awl ctr (JSR,ei\_awctr,icl awctr};   
where c\_awlctr is a routine as defined below:   
Function definition   
ec\_awctr (barpos,barlen,itempevent,chid,status,wman,subworkdefworkdef);   
Parameter declarations   
long wman,chidstatus,subworkdet,workdef;   
short item,barpos,barlen,\*pevent;   
Function return   
error code   
Parameter call return   
wman address of window manager vector base *0*barpos position of hit on bar *0*barlen length of the bar (pixels) *0*item pan/scroll item number (->previous page) *0*pevent address of \*event variable *0*\*pevent pan or scroll event number 0 or event to set   
chid channel ID *0*status pointer to status area *0*subworkdef pointer to sub-window working definition *0*workdef pointer to working definition *0*

**WM SETUP**Description   
This routine transfers the data from the menu window definition to the working definition. It is up to the application program to supply the memory where the working definition and status area are to be installed. Therefore the application program must calculate the size and allocate the memory, e.g. in a heap or within the data area of a job.   
The data for all loose menu items and information sub-windows are transferred completely. For each application sub-window the application sub-window setup routine is called from within WM\_SETUP. EASYMENU generated menu definitions always contain the code of a standard application sub-window setup routine for a non scrollable/pannable menu.   
This routine can be substituted by an own routine.   
See 3.1 for the details how to manage internal routines from C.   
Parameter declarations   
long err, ch id status ,wman wi ndef, workdef;   
short xsize,ysize   
Function call   
err = wm\_setup (chid,status,wman,windef,workdef,&xsize,&ysize);   
Function return   
err ->2.2   
Parameter call return   
wman pointer to window manager vector base 0   
chid channel ID if primary window or 0   
-1 for a secondary window   
&xsize address of xsize variable 0   
xsize x size \*) actual x size   
&ysize address of ysize variable 0   
ysize y size j actual y size   
status address of status area 0   
windef address of window definition 0   
workdef address of working definition 0   
\*) both 0 take default window size   
-1 keep (old) size unchanged   
4J find suitable layout and/or determine scaling factors   
..WM\_SETUP

...WMSETUP   
Assembler $04 wm.setup   
Register call return   
ID ? U   
D1.l dy size actual xly size   
**ftc** channel ID   
Al pointer to status area   
AS pointer to window definition   
M pointer to working definition   
From within WM\_SETUP, for each application sub-window, the application sub- window setup routine PI\_AWSET (wda\_setr) is called, but only if the pointer to this routine in the application sub-window definition is not zero.   
There are many possibilities what this routine could do, e.g.:   
- setting up a standard menu already defined in the window definition (this can be done by simply calling WM\_SMENU)   
- setting up a menu directly in the working definition, e.g. for dynamic purposes of the program.   
- altering a standard menu definition after it has been set up with WM\_SMENU, e.g change colours, set the pointers to the draw and control routines   
- prepare the window for any other usage   
Attention   
If a pannable and/or scrollable menu definition has been set up, also the section control block(s) must be set up. On a standard menu the EW\_AWSCT libraiy routine can be used, which is not a standard window manager routine. The window manager assumes the application sub-window setup routine to be written in assembler, i.e. the parameters are passed in the registers. To call a C routine, a small bit of assembler code is needed, to push the parameters on the stack. See 3.1 for more details.   
- **..WM\_SETUP**

ELAWSET   
Called from WMSETUP   
Assembler apptication nob-window -s®Rp routine   
Register call return   
DI.w x scaling   
D2.w y scaling   
Al pointer to status area 7   
42 window manager vector 7   
A3 pointer to sub-window definition ?   
*M* running pointer to working definition updated   
Description   
A3 points at the end of the base definition or at the end of the optional pan/scroll- control definitions, i.e to the start of the optional application sub-window menu definition (wda\_mstt).   
In the window definition, the x (y) control definition is only present, if the number of x (y) control sextions is not 0.   
The menu definition part is also optional.   
A4 is a running pointer to the working definition. On call it points at the end of the pan/scroll control definitions, which are always present in the working definition. On return A4 must point to the next free location in the working definition.   
Example:   
A C-written application sub-window setup routine can be called using an   
IN\_ROUT structure and the EI\_AWSET Library routine. See also 3.1.   
#define JSR 20153 /\* = $4EB9 = jsr /   
struct in\_rout   
short jump;   
long asm\_rout;   
long C\_funct;   
awl set ={JSR,ei\_awset,icl awset);   
If the above definitions and declarations are made outside the body of any function, the pointer to the application sub-window setup routine in the menu definition source code can be referred to as:   
xref awl set   
dc.w awlset-t pointer to setup routine   
**...WM\_SETUP**

...WF&SETUP   
Description   
An application sub-window routine written in C and called by help of an IN\_ROUT structure via EI\_AWSET must be defined as below.   
WMSMENU can be used as EC\_AWSET, as it requires exactly the same parameters, but it nothing else is to be done the simple assembler routine:   
section wman   
xdet ei\_awset   
eawset   
jmp wm.smenu(a2)   
end   
would do it.   
Parameter call return   
wman address of window manager vectors 0   
xscale x scaling factor 0   
yscale y scaling factor 0   
status address of status area 0   
psubwindef address of \*psubwindef variable 0   
\*psubwindet address of sub-window menu definition updated   
psubworkdef address of \*psubworkdet 0   
\*psubworkdef address of sub-window working definition updated   
Function return   
43 standard QDOS error code   
4) OK   
Function definition   
ec\_awset (xscale,yscale,status,wman,psubwindef,psubworkdef)   
shod xscaling,yscaling;   
long status,wman,\*psubwindef,\*psubworkdet;   
P Example: \*   
long em   
*I* now the routine itself, e.gjust make room for the application sub-window menu definition for later usage /   
\*psubworkdef\*psubworkdef+72;   
err=O;   
return(err);

**WM\_SMENU**Description -   
This routine transfers the data of a standard application sub-window menu definition from the window to the working definition.   
The routine is normally cailed from within the application sub-window setup routine ***(->*** EI\_AWSET).   
Parameter declarations   
long err, wman ,status,subwindef, su bworkdef;   
short xscaie,yscale;   
Function call   
err = wm\_smenu (xscale,yscale,status,wman,&subwindef,&subworkdef);   
Function return   
err ->2.2   
Parameter call return   
timeout ->2.4 *0*wman address of window manager vector base *0*status address status area *0*&subwindef address of subwindef variable *0*subwindef pointer to sub-window menu definition updated   
&subworkdef address of subworkdet variable *0*subworkdef running pointer to working definition updated to next free   
pointing to menu definition start in location in working   
application sub-window working definition definition   
Assembler *$* wm.smenu   
Register call return   
7 0   
D1.w xscaling   
D2.w y scaling   
Al pointer to status area =   
PS pointer to sub-window menu definition updated   
A4 pointer to working definition updated

**WM STIOB**Description   
This routine replaces an information object. Use WM\_IDPAW to redraw the information sub-window.   
Parameter declarations   
long errobject,wman,workdet   
short infonum,inobnum   
Function call   
err = wmstiob (inobnum,infonum,object,wman,workdef);   
Function return   
err ->2.2   
Parameter call return   
wman address of window manager vector base 0   
intonum information sub-window number 0   
inobnum information object number 0   
object address of new object 0   
workdef address of working definition 0   
Assembler $50 wm.stiob   
Register call return   
EU error code   
Dl info window! info object number   
Al pointer to new object   
P4 pointer to working definition   
Special error code meaning   
err.orng window or object number out of range

**WM\_STLOB   
Description   
This** routine replaces a loose menu item object. Use WM\_LDRAW to redraw the item.   
**Parameter declarations**long err,objectwman,workdef;   
short item   
**Function call**err = wm\_stlob (item,object,wman,workdet);   
**Function return**err ->22   
**Parameter call return**wman address of window manager vector base *0*item loose menu item number *0*object address of new object *0*workdef address of working definition 0   
Assembler $4C wm.stlob   
Register call return   
7 error code   
Dl item number   
Al pointer to new object   
*P4* pointer to working definition   
**Special error code meaning**err.orng item number out of range

**WM\_SWAPP   
Description   
This routine sets** the window area of the menu channel to the size of an application sub-window, if an ink colour is given, then also the ink colour is set, the paper and strip colours are set as defined for the window and the over status is set to 0.   
Parameter declarations   
long err,ink,wman,workdef;   
short winum;   
Function call   
err = wm\_swapp (winumink,wman,&subworkdef,workdef);   
Function return   
err ->2.2   
Parameter call return   
wman address of window manager vector base *0*winum application sub-window number *0*ink ink colour or -1 *0*-1 set area only   
>=0 set area and ink/paper/strip-colours *0*&subworkdef address of subworkdef variable *0*subworkdef ? address of sub-   
window working   
definition   
workdef address of working definition *0*Assembler $60 wm.swapp   
Register call return   
IX ? error code   
Di.w application sub-window number   
D2.I ink colour or-i   
AD channel ID   
A4 pointer to working definition   
Special error code meaning   
err.orng application sub-window number out of range

**WM\_SWDEF**Description -   
This routine sets the window area of the menu channel to the size and paper colour of an information or application sub-window.   
All other window attributes (over state, ink, strip colour, ..) are not affected.   
-> WM\_SWAPP, WM\_SWINF, WM\_SWLIT, WM\_SWSEC   
Parameter declarations   
long err, wman ,s ubwo rkdef,workdef;   
Function call   
err = wm\_swdef (wman,subworkdef,workdef);   
Function return   
err ->2.2   
Parameter call return   
wman address of window manager vector base *0*subworkdef address of sub-window definition *0*workdef address of working definition ***0***Assembler $28 wm.swdef   
Register call return   
ED error code   
D21 ink colour or-I =   
PD channel ID channel ID   
PS pointer to sub-window definition   
pointer to working definition

**WM\_SWDEF**Description   
This routine sets the window area of the menu channel to the size and paper colour of an information or application sub-window.   
All other window attributes (over state, ink, strip colour, ... ) are not affected.   
-> WM\_SWAPP, WM\_SWINF, WM\_SWLIT, WMSWSEC   
Parameter declarations   
long err,wman s ubwo rkdef wo rkdef;   
Function call   
err = wmswdef (wman,subworkdef,workdef);   
Function return   
err ->22   
Parameter call return   
wman address of window manager vector base ***0***subworkdef address of sub-window definition ***0***workdef address of working definition ***0***Assembler $28 wm.swdef   
Register call return   
II error code   
D2.l ink colour or-i   
channel ID channel ID   
PS pointer to sub-window definition   
P4 pointer to working definition

**WM SWLIT**Description   
This routine sets the window area of a menu channel to the size of a loose menu item. It the status is -1, only the window area is set, else the ink and paper colour is set according to the status and the over status is set to 0.   
Parameter declarations   
long err,wmanitmstat,workdet;   
short winum;   
Function call   
err = wm\_swlit (winurn,itmstatwman,&subworkdef,workdef);   
Function return   
err ->2.2   
Parameter call return   
wman address of window manager vector base *0*winum loose menu item number *0*itmstat -1 set area only or item status colours *0*&subworkdef address of subworkdet variable *0*subworkdef ? address of loose   
menu item working   
definition   
workdef address of working definition *0*Assembler $5C wm.swlit   
Register call return   
? error code   
D1.w loose menu item number   
D2.l item status   
channel ID   
P4 pointer to working definition   
Special error code meaning   
err.orng loose menu item number out of range   
-1 set area only   
0 = $00 set available colours   
16 = $10 set unavailable colours   
128 = $80 set selected colours

**WM SWSEC**Description   
This routine sets the window area of the menu channel to the size of an application sub-window menu section, if an ink colour is given, then also the ink, paper and strip colour are set and the over status is set to 0.   
Parameter declarations   
long err, in k,wman ,subwo rkdef,workdef;   
short xsect,ysect;   
Function call   
err = wm\_swsec (xsec,ysecink,wman,subworkdef,workdef);   
Function return   
err ->2.2   
Parameter call return   
wman address of window manager vector base *0*xsect **x** section number *0*ysect y section number *0*ink ink colour or-i 0   
-1 set area only   
>=0 set area and ink/paper/strip-colour 0   
subworkdef address of application sub-window definition 0   
workdet address of working definition 0   
Assembler $64 wmawsec   
Register call return   
W ? error code   
D1.w info window number   
02.1 ink colour or —1   
PD channel ID   
A4 pointer to working definition   
Special error code meaning   
err.orng application sub-window number or section number out of range

**WM UNSET**Description   
Clear pointer to working definition in extended channel definition block. If secondary window, restore area beneath and restore old pointer position (saved by WM\_PRPOS).   
In addition to the standard window manager routine, this library routine can also be used to unset a secondary window from the managed status, or to close the channel.   
Parameter declarations   
long err,wman,workdef;   
short flag;   
Function call   
err = wm\_unset (flag,wman,workdef);   
Function return   
er ->2.2   
Parameter call return   
wman address of window manager vector base 0   
flag 0 unset only 0   
1 unset and unmanage   
-1 close   
workdef pointer to working definition 0   
Assembler $14 wm.unset   
Register call return   
II error code   
PD ? channel ID   
M pointer to working definition   
Attention: The standard window manager routine does not close the window   
— channel!

**WM UPBAR**Description -   
This routine updates the parilscroll bars of an application sub-window menu section after a pan/scroll operation.   
Parameter declarations   
long err,wman ,subwo rkdet,workdef;   
short xsect,ysect;   
Function call   
err = wm\_upbar (xsect,ysect,wman,subworkdef,workdef);   
Function return   
err ->2.2   
Parameter call return   
wman address of window manager vector base *0*xsect x section number *0*ysect y section number *0*subworkdef address of sub-window definition *0*workdof address of working definition 0   
Assembler $70 wm.upbar   
Register call return   
DO.l x/y section number error code   
PD channel ID   
PS pointer to sub-window definition =   
M pointer to working definition =

**WM\_WDRAW**Description   
Draw entire menu window contents. The whole menu structure, information windows, looose menu items and application sub-windows are drawn. For any application sub-window, the application sub-window draw routine is called.   
Parameter declarations   
long err,wman, workdef;   
Function call   
S—. err = wm\_wdraw (wman,workdef);   
Function return   
err -> 2.2   
Parameter call return   
wman address of window manager vector base *0*workdef address of working definition *0*Assembler **$10** wm.wdraw   
Register call return   
IX ‘7 error code   
PD channel ID   
A4 pointer to working definition   
a TII   
From within WM\_WDRAW for each application sub-window the application sub- window draw routine EI\_AWDRW (wwa\_draw) is called, but only if the pointer to this routine in the application sub-window working definition is not zero. The pointer might have been transferred from the window definition or set in by the setup routine.   
— As for the application sub-window setup routine, the window manager assumes the routine to be written in assembler, thus a bit of assembler code is necessary to move parameters from the registers to the stack.   
.WM\_WDRAW

,WM\_WDRAW   
ELAWDRW   
Assembler appiicatbn bwdiow draw afliuie   
Register call return   
must be set   
ED 7 error code   
07.1 main window Wy origin   
AD channel ID   
Al 7   
pointer to window manager vector ?   
pointer to sub-window definition ?   
PA pointer to working detinition   
**.WM\_WDRAW**

..WM\_WDRAW   
ECAWDRW   
Description   
If a draw routine shall be written in C, an in\_rout structure must be used:   
struct in\_rout awidraw = {JSP,eLawdrw,iclawdrw};   
where iclawdrw is a routine as defined below:   
Parameter call return   
wman address od window manager vectors *0*xorg main window x pixel origin *0*yorg main window y pixel origin *0*chid channel ID *0*subworkdef address of sub-window definition *0*workdef address of working definition *0*Function definition   
ec\_awdrw (xorg,yorg,chid,wman,subworkdef,workdef)   
Parameter declarations   
short xorg,yorg;   
long chid,wman,subworkdefworkdef;   
long em   
/ e.g. the draw routine might call WM\_MDRAW and WM\_INDEX to draw a standard menu and pan/scroll bars /   
e rr=wm\_md raw(O, wman su bworkdef ,workdef);   
err=wmindex(wmansubworkdef,workdef);   
return(err);   
Function return   
error code

**WM WRSET   
Description   
Draw menu** window border and clear window. Set pointer to working definition in   
extended channel definition block   
This routine is called implicitely from WM\_PRPOS.   
Parameter declarations   
long errwman,workdef;   
Function call   
err = wm\_wrset (wman,workdet);   
Function return   
err -> 2.2   
Parameter call return   
wman address of window manager vector base *0*workdef address of working definition *0*Assembler $18 wm.wrset   
Register call return   
II ? error code   
? channel ID   
P4 pointer to working definition