Cortland Window Manager
Appendix A
Window Calls

WINDOW MANAGER ROUTINES

Initialization and Termination

WindBootInit *(not completed)*

input: None.

output: None.

Called only by **SetTSPtr**.

WindStartup

input: userID:WORD - user's ID that Window Manager can use.

output: None

WindStartup initializes the Window Manager. Calls the Event Manager for zero page to use, clears the window list, and sets the default desktop pattern and color. It creates the Window Manager port; you can get a pointer to this port with the **GetWMgrPort** procedure. The desktop is the entire screen until the Menu Manager, if used, subtracts any area needed for a system menu bar. Call this procedure once before all other Window Manager routines. **WindStartup** does not draw the desktop, see **Refresh**.

WindShutDown *(not completed)*

input: None.

output: None.

Frees any memory allocated by the Window Manager.

WindVersion

input: None.

output: wVersion:WORD - Window Manager's version number.

WindReset *(not completed)*

input: None.

output: None.

WindStatus *(not completed)*

input: None.

output: status:WORD -

WNewRes

input: None.

output: None.

Called after the screen resolution has been changed. The Window Manager will close its port and open a new one (in the new resolution). Then the screen is not redrawn by the Window Manager in the new resolution. Call Refresh when all resolution changes are done, such as desktop pattern and window colors.

Desktop

inputs: Operation:WORD - operation to perform/
Param:LONG - parameter needed for operation.

output: RetParam:LONG - possible return parameter.

Possible Operation numbers:

FromDesk	= 0	Subtract region from desktop region.
ToDesk	= 1	Add region to desktop region.
GetDesktop	= 2	Return handle of desktop region.
SetDesktop	= 3	Set handle of dektop region.
GetDeskPat	= 4	Return current desktop pattern.
SetDeskPat	= 5	Set new desktop pattern.
GetVisDesktop	= 6	Return desktop, less any windows.

Expected inputs and outputs:

Operation	= FromDesk
Param	= Handle of region to be subtracted from desktop region.
RetParam	= Not used (not even necessary to push room on stack).

Operation	= ToDesk
Param	= Handle of region to be added to desktop region.
RetParam	= Not used (not even necessary to push room on stack).

Operation	= GetDesktop
Param	= Not used.
RetParam	= Handle of desktop region.

Operation	= SetDesktop
Param	= Handle of new desktop region.
RetParam	= Same as Param.

Operation	= GetDeskPat
Param	= Not used.
RetParam	= Current desktop pattern where:

$00xxxxxx Where xxxxxx is the address of your routine that will be called to draw the desktop. There are no inputs or outputs,the current port will be the Window Manager's, and the clipping region will be set to the area needing to be drawn. Your routine should exit via a RTL.

Warning: The current direct page and data bank is not defined on entry to your routine. If you need to reference you own direct page you will have to save the original and switch to yours. The same is true with the data bank, except here you can use long addressing. When you exit your routine the direct page and data bank must be the same as it was on entry.

$80xxxxxx Where xxxxxx is the address of the pattern to be used for the desktop.

$4000xxxx The default desktop pattern where xxxx is:

00xx	= solid desktop pattern.
01xx	= dithered desktop pattern.
02xx	= horizontal stripped desktop pattern.
xxNx	= N is the pattern's foreground color.
xxxN	= N is the pattern's background color.

Operation	= SetDeskPat
Param	= New desktop pattern (see GetDeskPat for definition).
RetParam	= Not used (not even necessary to push room on stack).

Desktop is redrawn with new pattern.

Operation	= GetVisDesktop
Param	= Handle of region that will be set to the visible desktop.
RetParam	= Not used (not even necessary to push room on stack).

The desktop region is copied into the given region, and all visible windows are subtracted from it.

NewWindow

input: paramList:LONG - pointer to a parameter list.

output: theWindow:LONG - pointer to window port, zero if error.

Possible errors:
1 = incorrect parameter list length.
2 = unable to locate memory for window record.

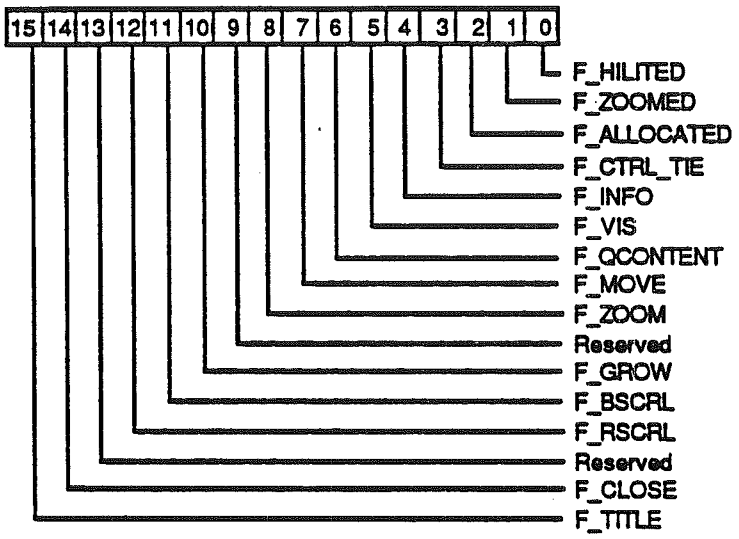
NewWindow creates a window as specified by its parameters, adds it to the window list, and returns a pointer to the new window's port. It allocates space for the structure and content regions of the window and asks the window definition function to calculate those regions.

The parameter list is:

param_length	WORD	Number of bytes in parameter table.
wFrame	WORD	Bit vector that descibes the window.
wTitle	LONG	Pointer to window's title.
wRefCon	LONG	Reserved for application's use only.
wZoom	RECT	Size and position of content when zoomed.
wColor	LONG	Pointer to window's color table.
wYOrigin	WORD	Content's vertical origin.
wXOrigin	WORD	Content's horizontal origin.
wDataH	WORD	Height of entire document.
wDataW	WORD	Width of entire document.
wMaxH	WORD	Maximum height of content allowed by GrowWindow.
wMaxW	WORD	Maximum width of content allowed by GrowWindow.
wScrollVer	WORD	Number of pixels to scroll content vertically for arrows.
wScrollHor	WORD	Number of pixels to scroll content horizontally for arrows.
wPageVer	WORD	Number of pixels to scroll content vertically for page.
wPageHor	WORD	Number of pixels to scroll content horizontally for page.
wInfoRefCon	LONG	Value passed to information bar draw routine.
wFrameDefProc	LONG	Address of standard window definition procedure.
wInfoDefProc	LONG	Address of routine that draw's the information bar interior.
wContDefProc	LONG	Address of routine that draw's the content region interior.
wPosition	RECT	Window's starting position and size.
wPlane	LONG	Window's starting plane.
wStorage	LONG	Address of memory to use for window record.

Each parameter is covered in more detail below.

param_length	Total number of bytes in parameter table, including param_length. Use labels in code to come up with the value (that's why I don't give it here). The value is used mainly for error checking. Most errors with NewWindow occur because of typing errors when creating the parameter list. The problem can be compounded further by the assembler or complier skipping field because of typing errors but not generating an error.
wFrame	Window frame type:



F_HILITED	1 = window is highlited, 0 = not highlighted. This flag will be set by NewWindow, so whatever you pass will be ignored.
F_ZOOMED	1 = window is currently in a zoomed state, 0 = window is not zoomed. This flag is not used is F_ZOOM is zero.
F_ALLOCATED	1 = window record was allocated by the NewWindow, 0 = window record was not allocated by the NewWindow. If this flag is set when CloseWindow is called, the window record will be freed.
F_CTRL_TIE	1 = the state of the window's controls is not tied to the window's state. 0 = when the window is inactive (unhighlighted), its controls are also considered inactive without regard for the active state of the control.
F_INFO	1 = window has an information bar as part of the window's frame, 0 = no information bar.
F_VIS	1 = window is visible, 0 = window is invisible.

F_QCONTENT	1 = if there is a button down event inside an inactive window's content, the window will be selected and a wInContent message will be returned by TaskMaster. This feature is use if you would like to act on any button down in the content, even if it was also used to activate the window. If this bit is zero TaskMaster will act in the same way, execpt it will return an inNull message. This feature is use if you would like to button down in an inactive content to activate the window and then not use the same button down event again.
F_MOVE	1 = window can be dragged by its title bar, 0 = the window's title bar is not considered a drag region and can not, therefore, be moved.
F_ZOOM	1 = window has a zoom box in its title bar, 0 = window does not have a zoom box in its title bar. The window must have a title bar in order to have a zoom box.
F_GROW	1 = window has a grow box, 0 = window does not have a grow box.
F_BSCRL	1 = window has a bottom (horizontal) scroll bar as part of the window frame, 0 = no bottom scroll bar.
F_RSCRL	1 = window has a right (vertical) scroll bar as part of the window frame, 0 = no right scroll bar.
F_CLOSE	1 = window has a close box in its title bar, 0 = window does not have a close box in its title bar. The window must have a title bar in order to have a close box.
F_TITLE	1 = window has a title bar as part of the window's frame, 0 = no title bar.

Warning: If F_GROW is set, F_BSCRL or F_RSCRL must also be set. That is to say, to have a window frame grow box, you must have at least one window frame scroll bar.

wTitle	Pointer to title of window. If window does not have a title bar this value can be zero. The first byte in the string should be the length of the string, followed byte the ASCII characters of the title.
wRefCon	Application defined reference value. This value is reserved for the application's use only, and can be any value desired.

wZoom	Rectangle of the content region when the window is zoomed. If the bottom side of the rectangle is zero, a default RECT will be used. The default will be set so that the window will use the entire screen.
wColor	Pointer to window's color table. This is the color table used to draw the window's frame. Zero to use the default color table.
wYOrigin	Vertical offset of content region. This value is the vertical value passed to **SetOrgin** when **TaskMaster** is used to draw inside the content region. It is also used to compute the right (or vertical) scroll bar. Zero if not using window frame scroll bars.
wXOrigin	Horizontal offset of content region. This value is the horizontal value passed to **SetOrgin** when **TaskMaster** is used to draw inside the content region. It is also used to compute the bottom (or horizontal) scroll bar. Zero if not using window frame scroll bars.
wDataH	Height of entire data area. Used to compute the right scroll bar. Zero if not using window frame scroll bars.
wDataW	Width of entire data area. Used to compute the bottom scroll bar. Zero if not using window frame scroll bars.
wMaxH	Maximum content height allowed when growing the window. This value is passed to **GrowWindow** when called by **TaskMaster.** If set to zero, a default value will be used, so that the window will take up the height of the desktop. Zero if not using window frame grow box.
wMaxW	Maximum content width allowed when growing the window. This value is passed to **GrowWindow** when called by **TaskMaster.** If set to zero, a default value will be used, so that the window will take up the width of the desktop. Zero if not using window frame grow box.
wScrollVer	Number of pixels to scroll the content region when the up or down arrows are selected in the right scroll bar. Used only if the scroll bar is part of the frame and **TaskMaster** is used. Zero if not using window frame scroll bars.
wScrollHor	Number of pixels to scroll the content region when the left or right arrows are selected in the bottom scroll bar. Used only if the scroll bar is part of the frame and **TaskMaster** is used. Zero if not using window frame scroll bars.
wPageVer	Number of pixels to scroll the content region when the up or down page regions are selected in the right scroll bar. Used only if the scroll bar is part of the frame and **TaskMaster** is used. Zero if not using window frame scroll bars.
wPageHor	Number of pixels to scroll the content region when the left or right page regions are selected in the bottom scroll bar. Used only if the scroll bar is part of the frame and **TaskMaster** is used. Zero if not using window frame scroll bars.

wInfoRefCon	Value passed to Information Bar draw routine. The value can be anything the application would like, such as a pointer to a string to be printed in the information bar. Zero if not using window frame information bar.
wFrameDefProc	Pointer to window's definition procedure. Zero for a standard document window.
wInfoDefProc	Address of routine that will be called to draw in the information bar. Zero if not using window frame information bar.
wContDefProc	Address of routine that will be called to draw the window's content region. If you are using window frame scroll bars this value must be set. If you are not using window frame scroll bars this value can be zero. However, if you are not using window frame scroll bars, but you would like TaskMaster to handle update events, set this value. The routine will be called when the content region needs to be drawn. On entry, the current port will be the window's, the visible region region will be set to the update area, and the origin set. There are no input or output parameters. Exit the routine via RTL.
wPosition	A rectangle given in global coordinates, determines the window's size and location, and becomes the portRect of the window's grafPort; note, however, that the portRect is in local coordinates. **NewWindow** sets the top left corner of the portRect to (0,0). For the standard types of windows, this RECT defines the content region of the window.
wPlane	Pointer to window port this window should appear behind. Zero for bottom most, $FFFFFFFF for top most.
wStorage	Address of memory to use for window's record. If set to zero, the record will be allocated. Because window records are not completely defined, the size needed for a window record is unknow. Therefore, you must allow 325 bytes for a window record. Actually, it is best to have the record allocated by the Window Manager. Being able to use your own memory for a window record is provided for in case you need to put up a window to say there is no memory left, and therefore the Window Manager could not allocate one.

Note: The bit map, pen pattern, and other characteristics of the window's grafPort are the same as the default values set by the **OpenPort** procedure in QuickDraw. (**NewWindow** actually calls **OpenPort** to initialize the window's grafPort.) Note, however, that the coordinates of the grafPort's portBits.bounds and visRgn are changed along with its portRect.

NewWindow also sets the window class in the window record to indicate that the window was created directly by the application.

CloseWindow

input: theWindow:LONG - pointer to window's port.

output: None.

CloseWindow removes the given window from the screen and deletes it from the window list. It releases the memory occupied by all data structures associated with the window, including the memory taken up by the window record if it was allocated by **NewWindow**. Call this procedure when you're done with a window.

Any update events for the window are discarded. If the window was the frontmost window and there was another window behind it, the latter window is highlighted and an appropriate activate event is generated.

Warning: If you allocated memory yourself and stored a handle to it in the refCon field, **CloseWindow** won't know about it—you must release the memory before calling **CloseWindow**.

Window Record and Global Access

GetWMgrPort

input: None.

output: wPort:LONG - pointer to Window Manager's port.

SetWMgrIcons

input: NewFont:LONG - handle of new icon font to use, negative to not replace font.

output: OldFont:LONG - handle of icon font before replacement, if any.

See WINDOW MANAGER ICON FONT for more information about the font.

SetWRefCon

inputs: refCon:LONG - reserved LONG for application's use
theWindow:LONG - pointer to window's port.

output: None.

SetWRefCon is used to set a LONG value that is inside the window record and is reservered for the application's use.

GetWRefCon

input: theWindow:LONG - pointer to window's port.

output: refCon:LONG - reserved LONG for application's use

GetWRefCon is used to retrieve a LONG value from a window's record that was passed to either **NewWindow** or **SetWRefCon** by the application sometime before this call.

SetWTitle

inputs: title:LONG - pointer to string for new title.
theWindow:LONG - pointer to window's port.

output: None.

Updates window's record with new title pointer

GetWTitle

input: theWindow:LONG - pointer to window's port.

output: title:LONG - pointer to string of window's title.

SetFrameColor

inputs: newColor:LONG - pointer to 8 word pattern/color table, zero for system.
theWindow:LONG - pointer to window's port, zero to set default.

output: None.

See WINDOW FRAME COLORS AND PATTERNS for a definition of the color table. Does not redraw the window. Do a HideWindow and ShowWindow before and after this call to redraw the window in its new colors.

If newColor is zero, the pointer to the system default color table will be used. If theWindow is zero, the default window color table will be set. To understand defaults, system, and all this, it necessary to understand how the Window Manager finds a color table to use for drawing. First, a field in the window record is checked for a pointer to a color table. The field is zero after allocated by **NewWindow**, and remains zero until a **SetFrameColor**. If a pointer is found in the window's record, that is the table used. If a zero is found, the default table is used. Now comes the tricky part, the default table starts out as the system table, but can be changed by **SetFrameColor** when theWindow is zero.

GetFrameColor

inputs: newColor:LONG - pointer to 8 word table that will be set with the color table.
theWindow:LONG - pointer to window's port.

output: None.

See WINDOW FRAME COLORS AND PATTERNS for a definition of the color table.

FrontWindow

input: None.

output: theWindow:LONG - pointer to the active window's port.

FrontWindow returns a pointer to first visible window in the window list (that is, the active window). If there are no visible windows, it returns zero.

GetNextWindow

input: theWindow:LONG - pointer to window's port.

output: NextWindow:LONG - pointer to next window's port in list, zero is last.

GetNextWindow returns a pointer the next window after theWindow in the window list, or zero if theWindow is the last window in the window list.

GetWKind

input: theWindow:LONG - pointer to window's port.

output: WindowKind:WORD - TRUE if system window, FALSE if application window.

GetWKind returns the kind of window theWindow is.

GetWFrame

input: theWindow:LONG - pointer to window's port.

output: wFlag:WORD - bit vector of window's frame type.

GetWFrame returns the same type of bit vector passed to NewWindow. See NewWindow for the definition of the bits of wFlag.

SetWFrame

input: wFlag:WORD - bit vector of window's frame type.
theWindow:LONG - pointer to window's port.

output: None.

SetWFrame sets the same type of bit vector passed to NewWindow. See NewWindow for the definition of the bits of wFlag. The window frame is not redrawn.

GetStructRgn

input: theWindow:LONG - pointer to window's port.

output: WStructRgn:LONG - handle of window's structure region.

See WINDOW REGIONS for a definition of what the structure region is.

GetContRgn

input: theWindow:LONG - pointer to window's port.

output: WContRgn:LONG - handle of window's content region.

See WINDOW REGIONS for a definition of what the content region is.

GetUpdateRgn

input: theWindow:LONG - pointer to window's port.

output: WUpdateRgn:LONG - handle of window's structure region.

See BeginUpdate for an explaination of how the update region is used.

GetDefProc

input: theWindow:LONG - pointer to window's port.

output: WDefProc:LONG - pointer theWindow's definition procedure.

GetDefProc returns the address of the routine that is called to draw, hit test, and otherwise define, a window's frame and behavior.

SetDefProc

input: WDefProc:LONG - pointer theWindow's definition procedure.
theWindow:LONG - pointer to window's port.

output: None.

SetDefProc sets the address of the routine that is called to draw, hit test, and otherwise define, a window's frame and behavior. See DEFINING YOUR OWN WINDOWS for an explaination of what a definition procedure does.

GetWControls

input: theWindow:LONG - pointer to window's port.

output: ControlList:LONG - address of first control in window's control list, zero = none.

GetWControl returns the address of the first control in the window's control list. The window's control list is the list of controls created by the application with calls to **NewControl** in the Control Manager. The window's control list is separate from the window frame's control list, explained in **GetFControl.**

GetFControls

input: theWindow:LONG - pointer to window's port.

output: FControlList:LONG - address of first control in window's frame, zero = none.

GetFControls returns the address of the first control in the window frame's control list. The window frame's control list is the list of controls created by the window definition procedure for standard window controls like scroll bars.

GetInfoText

input: theWindow:LONG - pointer to window's port.

output: InfoText:LONG - value that will be passed to the information bar draw routine.

See INFORMATION BAR DRAW ROUTINE.

SetInfoText

input: InfoText:LONG - value that will be passed to the information bar draw routine.
theWindow:LONG - pointer to window's port.

output: None.

See INFORMATION BAR DRAW ROUTINE.

GetFullRect

input: theWindow:LONG - pointer to window's port.

output: wFullSize:LONG - pointer to RECT to be used as content's zoomed size.

If the zoom flag is set in the frame flag, see **GetWFrame**, then wFullSize will equal theWindow's last size and position. Otherwise, wFullSize will equal the size and position of theWindow's content region (port) the next time the window is zoomed via a call to **ZoomWindow**.

SetFullRect

input: wFullSize:LONG - pointer to RECT to be used as content's zoomed size.
theWindow:LONG - pointer to window's port.

output: None.

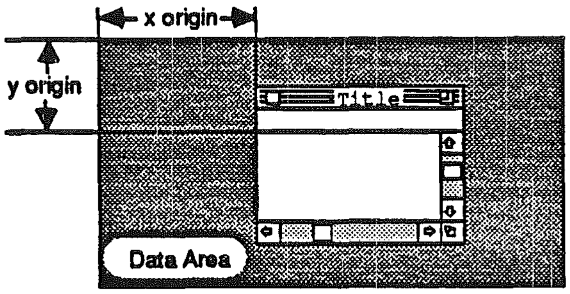
If the zoom flag is set in the frame flag, see **GetWFrame**, then wFullSize will equal theWindow's last size and position. Otherwise, wFullSize will equal the size and position of theWindow's content region (port) the next time the window is zoomed via a call to **ZoomWindow**.

GetCOrgin

inputs: theWindow:LONG - pointer to window's port.

output: LONG - low WORD = y origin, high WORD = x origin.

These values are used by **TaskMaster** for setting the origin of the window's port when handling an update event. The values are also used to compute scroll bars in the window frame.



SetCOrigin

inputs: xOrigin:WORD - content region's horizontal offset into the data area.
yOrigin:WORD - content region's vertical offset into the data area.
theWindow:LONG - pointer to window's port.

output: None.

See **GetCOrigin** for a description of origins. Setting these values will not generate any update event, although the entire content will probably needed to be redrawn.

SetOrgnMask

inputs: originMask:WORD - mask used to put horizontal origin on a grid.
theWindow:LONG - pointer to window's port.

output: None.

SetOrgnMask is useful when you are using a scrollable window in 640 mode with dithered colors. The video hardware of the Cortland is such that different pixel position get their color from different color tables. By using the effect it is possible to produce many more colors than the two bits per pixels might suggest. However, the pixels are then horizontally position dependent to keep the same color. Scrolling windows can change the color by putting the pixels in the wrong horizontal position. That's where SetOrgnMask comes in. OriginMask will be ANDed by TaskMaster with any new horizontal origin that is created to force the origin to certain boundaries. The default is $FFFF, single pixel.

StartDrawing

input: theWindow:LONG - pointer to window's port.

output: None.

StartDrawing can be used for drawing in a window's content region outside of update events. **StartDrawing** will make the window the current port, and set its origin. After the call, any drawing, outside of update events, will occur inside theWindow's content and in the proper coordinate system.

> **Note:** **StartDrawing** is only of use with standard document window's with frame scroll bars. Otherwise, only a **SetPort** would be needed to make the proper port current.

GetDataSize

inputs: theWindow:LONG - pointer to window's port.

output: dataSize:LONG - low WORD is the height, high WORD is the width.

The height and width of the data area is returned. The data area is the total amount of data that can be viewed in a window, either through resizing or scrolling.

SetDataSize

inputs: dataWidth:WORD - width of data area.
dataHeight:WORD - height of data area.
theWindow:LONG - pointer to window's port.

output: None.

See **GetDataSize**. Setting these values will not change the scroll bars or generate update events.

GetMaxGrow

inputs: theWindow:LONG - pointer to window's port.

output: maxGrow:LONG - low WORD is the max height, high WORD is the max width.

These values are pasted to **GrowWindow** by **TaskMaster**. The content region will not be allowed to be sized to exceed these values.

SetMaxGrow

inputs: maxWidth:WORD - maximum content width allowed when resizing.
maxHeight:WORD - maximum content height allowed when resizing.
theWindow:LONG - pointer to window's port.

output: None.

See GetMaxGrow.

GetScroll

inputs: theWindow:LONG - pointer to window's port.

output: scroll:LONG - low WORD is the vertical amount, high WORD the horizontal.

Returns the number of pixels that TaskMaster will scroll the content region when the user selects the arrows on window frame scroll bars.

SetScroll

inputs: hScroll:WORD - number of pixels to scroll horizontally.
vScroll:WORD - number of pixels to scroll vertically.
theWindow:LONG - pointer to window's port.

output: None.

See GetScroll.

GetPage

inputs: theWindow:LONG - pointer to window's port.

output: page:LONG - low WORD is the vertical amount, high WORD the horizontal.

Returns the number of pixels that TaskMaster will scroll the content region when the user selects the page regions on window frame scroll bars.

SetPage

inputs: hPage:WORD - number of pixels to page vertically.
vPage:WORD - number of pixels to page horizontally.
theWindow:LONG - pointer to window's port.

output: None.

See GetPage.

GetCDraw

inputs: theWindow:LONG - pointer to window's port.

output: contDraw:LONG - address of routine that is called to draw the content region.

TaskMaster will call this routine when it gets an update event for that window. See CONTENT DRAW ROUTINE for more infomation about the draw routine.

SetCDraw

inputs: contDraw:LONG - address of routine to draw content region.
theWindow:LONG - pointer to window's port.

output: None.

See GetCDraw.

GetInfoDraw

inputs: theWindow:LONG - pointer to window's port.

output: InfoDraw:LONG - address of routine that will draw on the infomation bar.

The standard window definition procedure will call this routine whenever the window's frame needs to be draw, if the window has an infomation bar. See INFORMATION BAR DRAW ROUTINE for more infomation about the draw routine.

SetInfoDraw

inputs: InfoDraw:LONG - address of routine that will draw on the information bar.
theWindow:LONG - pointer to window's port.

output: None.

See **GetInfoDraw.**

Window Shuffling

SelectWindow

input: theWindow:LONG - pointer to window's port.

output: None.

SelectWindow makes theWindow the active window as follows: It unhighlights the previously active window, brings theWindow in front of all other windows, highlights theWindow, and generates the appropriate activate events. Call this procedure if you are not using **TaskMaster** and there's a mouse-down event in the content region of an inactive window.

HideWindow

input: theWindow:LONG - pointer to window's port.

ouput: None.

HideWindow makes theWindow invisible. If theWindow is the frontmost window and there's a window behind it, **HideWindow** also unhighlights theWindow, brings the window behind it to the front, highlights that window, and generates appropriate activate events. If theWindow is already invisible, HideWindow has no no effect.

ShowWindow

inputs: theWindow:LONG - pointer to window's port.

output: None.

Makes theWindow visible and draws it if it was invisible. It does not change the front-to-back ordering of the windows. Remember that if you previously hid the frontmost window with **HideWindow**, **HideWindow** will have brought the window behind it to the front; so if you then do a **ShowWindow** of the window you hid, it will no longer be frontmost. If theWindow is already visible, **ShowWindow** has no effect.

ShowHide

input: showFlag:WORD - TRUE to show, FALSE to hide.
theWindow:LONG - pointer to the window's port.
ouput: None.

If showFlag is TRUE, **ShowHide** makes theWindow visible if it's not already visible and has no efect if it is already visible. If showFlag is FALSE, **ShowHide** makes theWindow invisible if it's not already invisible and has no effect if it is already invisible. Unlike **HideWindow** and **ShowWindow**, **ShowHide** never changes the highlighting or front-to-back ordering of windows or generates activate events.

> **Warning:** Use this procedure carefully, and only in special circumstances where you need more control than allowed by **ShowWindow** and **HideWindow**. You could end up with an active window that isn't highlighted.

BringToFront

input: theWindow:LONG - pointer to window's port.

output: None.

BringToFront brings theWindow to the front of all other windows and redraws the windows as necessary, but does not do any highlighting or unhighlighting. Normally you won't have to call this procedure, since you should call **SelectWindow** to make a window active, and **SelectWindow** takes care of bringing the window to the front. If you do call **BringToFront**, however, remember to call **HiliteWindow** to make the necessary highlighting changes.

SendBehind

inputs: behindWindow:LONG - pointer to window record or -2 to send to bottom.
theWindow:LONG - pointer to window's port.

output: None.

SendBehind sends theWindow behind behindWindow, redrawing any exposed windows. If behindWindow is -2 ($FFFFFFFE), it sends theWindow behind all other windows. If theWindow is the active window, it unhighlights theWindow, highlights the new active window, and generates the appropriate activate events.

Window Drawing

HiliteWindow

input: fHilite:WORD - TRUE to highlight window frame, FALSE to unhighlight.
theWindow:LONG - pointer to window's port.

output: None.

If fHilite is TRUE, this procedure highlights theWindow. If fHilite is FALSE, **HiliteWindow** unhighlights theWindow. The exact way a window is highlighted and unhighlighted depends on its window definition procedure.

Normally you won't have to call this procedure, since you should call **SelectWindow** to make a window active, and **SelectWindow** takes care of the necessary highlighting changes. Highlighting a window that isn't the active window should never be done.

Refresh

input: None.

output: None.

Redraws the entire desktop and all the windows. Useful when the entire screen was clobblered by some application specific, non-Window Manager, operation.

User Interaction

FindWindow

inputs: whichWindow:LONG - address of where to store pointer of window.
pointX - x coordinate to check (global).
pointY - y coordinate to check (global).

outputs: Location:WORD:

When a mouse-down event occurs, the application should, if not using **TaskMaster**, call **FindWindow** with pointY,pointX equal to the point where the mouse button was pressed (in global coordinates, as stored in the where field of the event record). **FindWindow** tells which part of which window, if any, the mouse button was pressed in. If it was pressed in a window, theWindow parameter is set to the window port pointer; otherwise, it's set to zero. The WORD returned by **FindWindow** is one of the following predefined constants:

wNoHit	= $0000	Not on the window at all.
wInDesk	= $0010	On the desktop area.
wInMenuBar	= $0011	On the system menu bar.
wInContent	= $0013	In window's content region.
wInDrag	= $0014	In window's drag (title bar) region.
wInGrow	= $0015	In window's grow (size box) region.
wInGoAway	= $0016	In window's go-away (close box) region.
wInZoom	= $0017	In window's zoom (zoom box) region.
wInInfo	= $0018	In window's information bar.
wInFrame	= $001B	In window, but not any of the above areas.
wInSysWindow	= $8xxx	In a system window, lower part is one of the above.

DragWindow

inputs: grid:WORD - drag resolution, zero for default.
startX - starting x coordinate of cursor (global).
startY - starting y coordinate of cursor (global).
grace:WORD - grace buffer around Bounds.
BoundsRect:LONG - pointer to RECT to use as cursor boundary, zero for default.
theWindow:LONG - pointer to window's port.

output: None.

When there is a mouse-down event in the drag region of theWindow, and TaskMaster is not being used, the application should call **DragWindow** with startY,startX equal to the point where the mouse button was pressed (in global coordinates, as stored in the where field of the event record). **DragWindow** pulls a dotted outline of theWindow around, following the movements of the mouse until the button is released. When the mouse button is released, **DragWindow** call **MoveWindow** to move theWindow to the location to which it was dragged. The window will be dragged and moved in its current plane..

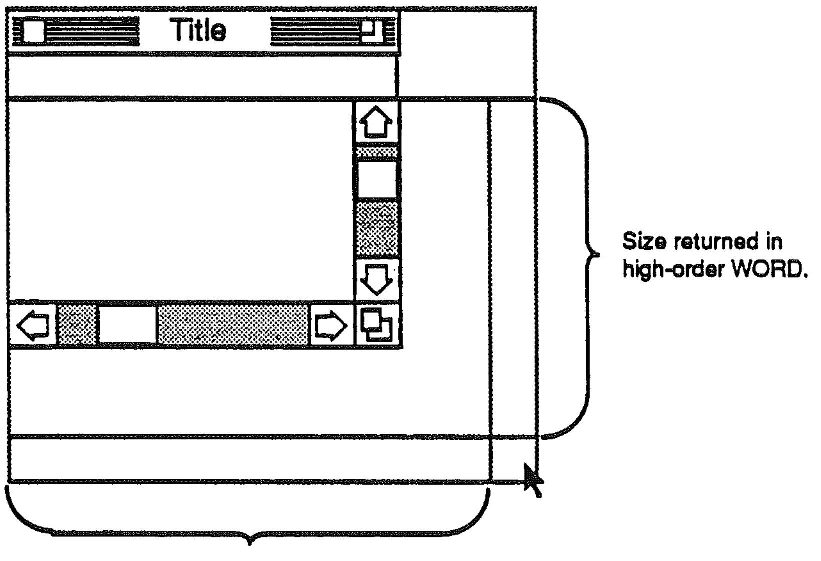
grid	Allowed horizontal resolution movement. If grid is one, the window can be positioned at any horizontal position. If grid is two, the window can only be moved a multiple of 2 pixels horizontally. If grid is four, the window can only be moved a multiple of 4 pixels horizontally. The only allowed values are; 1, 2, 4, 8, 16, 32, 64, 128... The grid parameter is provided to speed up window moves by eliminating the need for bit shifting, if the grid value is the correct value. If grid is passed as zero, a default value will be used. The defaults are; 4 for 320 mode and 8 for 640 mode.
startY,startX	The point where the mouse button was pressed, in global coordinates, as stored in the where field of the event record. This point is used with the tracked cursor position to compute the movement delta.
grace	Grace is the distance, in pixels, that you will allow the cursor to move away from BoundsRect before the dragged outline should be snapped back to its starting position. **TaskMaster** uses 8 for this value. The BoundsRect is expanded by the value of grace to compute the slopRect passed to **DragRect**. See **DragRect** for more information.
BoundsRect	Pointer to a RECT, in global coordinates, that is passed to **DragRect** as the limitRect parameter. See **DragRect** for more information. If zero is passed for the pointer, the bounds of the desktop, less 4 all around, will be used.

GrowWindow

inputs: minWidth:WORD - minimum width of content region to allow.
minHeight:WORD - minimum height of content region to allow.
startX - starting x coordinate of cursor (global).
startY - starting y coordinate of cursor (global).
theWindow:LONG - pointer to window's port.

output: newSize:LONG - high WORD = new height, low WORD = new width.

When there's a mouse-down event in the grow region of theWindow, the application should call **GrowWindow** with startY,startX equal to the point where the mouse button was pressed (in global coordinates, as stored in the where field of the event record). **GrowWindow** pulls a grow image of the window around, following the movements of the mouse until the button is released. The grow image for a document window is a dotted outline of the entire window and also the lines delimiting the title bar, size box, and scroll bar areas. The diagram below illustrates this for a document window containing both scroll bars. In general, the grow image is defined in the window definition function and is whatever is appropriate to show that the window's size will change.



Size returned in low-order WORD.

The application should subsequently call **SizeWindow** to change the portRect of the window's grafPort to the new one outlined by the grow image. The sizeRect parameter specifies limits, in pixels, on the vertical and horizontal measurements of what will be the new portRect. SizeRect.top is the minimum vertical measurement, sizeRect.left is the minimum horizontal measurement, sizeRect.bottom is the maximum vertical measurement, and sizeRect.right is the maximum horizontal measurement.

GrowWindow returns the actual size for the new portRect as outlined by the grow image when the mouse button is released. The high-order WORD of the LONG is the vertical measurement in pixels and the low-order WORD is the horizontal measurement. A return value of zero indicates that the size is the same as that of the current portRect.

TrackGoAway

inputs: startX - starting x coordinate of cursor (global).
startY - starting y coordinate of cursor (global).
theWindow:LONG - pointer to window's port.

output: GoAway:WORD - TRUE if go away selected when button released, else FALSE.

When there's a mouse-down event in the go-away region of theWindow, and the application is not using **TaskMaster**, the application should call **TrackGoAway** with thePT equal to the point where the mouse button was pressed (in global coordinates, as stored in the where field of the event record). **TrackGoAway** keeps control until the mouse button is released, highlighting the go-away region as long as the mouse location remains inside it, and unhighlighting it when the mouse moves outside it. The exact way a window's go-away region is highlighted depends on its window definition procedure. If the mouse button is released inside the go-away region, **TrackGoAway** unhighlights the go-away region and returns TRUE (the application should then eventually perform a **CloseWindow**). If the mouse button is released outside the go-away region, **TrackGoAway** returns FALSE (in which case the application should do nothing).

TrackZoom

inputs: startX - starting x coordinate of cursor (global).
startY - starting y coordinate of cursor (global).
theWindow:LONG - pointer to window's port.

output: Zoom:WORD - TRUE if zoom region was selected, else FALSE.

When there's a mouse-down event in the zoom region of theWindow, and the application is not using **TaskMaster**, the application should call **TrackZoom** with thePT equal to the point where the mouse button was pressed (in global coordinates, as stored in the where field of the event record). **TrackZoom** keeps control until the mouse button is released, highlighting the zoom region as long as the mouse location remains inside it, and unhighlighting it when the mouse moves outside it. The exact way a window's zoom region is highlighted depends on its window definition procedure. If the mouse button is released inside the zoom region, **TrackZoom** unhighlights the zoom region and returns TRUE (the application should then eventually perform a **ZoomWindow**). If the mouse button is released outside the zoom region, **TrackZoom** returns FALSE (in which case the application should do nothing).

TaskMaster

input: EventMask:WORD - used to call GetNextEvent.
TaskRec:LONG - pointer to an extended event record to use.

output: TaskCode:WORD - task code, zero equal no further task to perform.

Possible error: 3 = bits 12-15 are not clear in TaskMask field of TaskRec.

See USING TASKMASTER for more information.

TaskMaster uses TaskRec and EventMask to pass to GetNextEvent. An outline of TaskMaster follows:

```
Call SystemTask for possible desk accessories.
Call GetNextEvent with a TaskRec and EventMask.
If GetNextEvent returns 'no event' TaskMaster will exit and return inNull.
The message field of the TaskRec is duplicated into the TaskData field.

If event code is key down event:
    If TaskMask bit #0 = 0:
        TaskMaster exits and returns inKey.
    Call MenuKey for the system menu bar with the key from TaskRec.
    If MenuKey returns 'no selection made' TaskMaster exits and returns inKey.
    If TaskMask bit #4 = 0:
        TaskMaster exits and returns wInMenuBar.
    If the item selected has an ID number greater than 255:
        TaskMaster exits and returns wInMenuBar.
    Else the item belongs to a desk accessory and:
        Call OpenNDA to open the desk accessory selected.
        Call HiliteMenu to unhighlight the selected menu.
        TaskMaster exits and returns inNull.

If event code is update event:
    If TaskMask bit #1 = 0:
        TaskMaster exits and returns inUpdate.
    If the window with the update has an update draw routine (see NewWindow):
        Switch to window's port.
        Window's origin is set according to the origin values in its record.
        The window's update draw routine is called (routine in application).
        Window's origin is returned to zero,zero.
        The previous port is restored.
        TaskMaster exits and returns inNull.
    Else TaskMaster is unable to process the event:
        TaskMaster exits and returns inUpdate.
```

```
If event code is button down event:
    If TaskMask bit #2 = 0:
        TaskMaster exits and returns inButtDwn.
    Call FindWindow which place any found window pointer in the TaskData field.

    If FindWindow  returns wInMenuBar:
        If TaskMask bit #3 = 0:
            Low-order WORD of TaskData field in TaskRec = zero.
            TaskMaster exits and returns wInMenuBar.
        Call MenuSelect.
        If MenuSelect returns 'no selection made':
            TaskMaster exits and returns inKey.
        Else if the item selected ID number greater than 255 OR TaskMask bit #4 = 0:
            Low-order WORD of TaskData field in TaskRec = selected item's ID.
            High-order WORD of TaskData field in TaskRec = selected menu's ID.
            TaskMaster exits and returns wInMenuBar.
        Else the item belongs to a desk accessory and:
            Call OpenNDA to open the desk accessory selected.
            Call HiliteMenu to unhighlight the selected menu.
            TaskMaster exits and returns inNull.

    Else if FindWindow returns a value that is negative:
        If TaskMask bit #5 = 0:
            TaskData = window pointer returned from FindWindow.
            TaskMaster exits and returns value returned by FindWindow.
        FindWindow found something in a system window.
        Call SystemClick with the window and result from FindWindow.
        NOTE: This is as far as system windows can go in TaskMaster.
        TaskMaster exits and returns inNull.

    Else if FindWindow returns wInDrag:
        If TaskMask bit #6 = 0:
            TaskData = window pointer returned from FindWindow.
            TaskMaster exits and returns wInDrag.
        If the command key is not down and the window is not active:
            Call SelectWindow to make the window active.
        Call DragWindow.
        TaskMaster exits and returns inNull.

    Else if FindWindow returns wInContent:
        If TaskMask bit #7 = 0:
            TaskData = window pointer returned from FindWindow.
            TaskMaster exits and returns wInContent.
        If the window is not active:
            Call SelectWindow to make the window active.
            TaskMaster exits and returns inNull.
        Else:
            TaskData = window pointer returned from FindWindow.
            TaskMaster exits and returns wInContent.
    Else if FindWindow returns wInGoAway:
        If TaskMask bit #8 = 0:
            TaskData = window pointer returned from FindWindow.
            TaskMaster exits and returns wInGoAway.
        Call TrackGoAway.
        If TrackGoAway returns TRUE:
            TaskData = window pointer returned from FindWindow.
            TaskMaster exits and returns wInGoAway.
        TaskMaster exits and returns inNull.

    Else if FindWindow returns wInZoom:
        If TaskMask bit #9 = 0:
            TaskData = window pointer returned from FindWindow.
            TaskMaster exits and returns wInZoom.
        Call TrackZoom.
        If TrackZoom returns TRUE:
            Call ZoomWindow.
        TaskMaster exits and returns inNull.

    Else if FindWindow returns wInGrow:
        If TaskMask bit #10 = 0:
            TaskData = window pointer returned from FindWindow.
            TaskMaster exits and returns wInGrow.
        Call GrowWindow.
        Call SizeWindow with results from GrowWindow.
        TaskMaster exits and returns inNull.

    Else if FindWindow returns wInFrame:
        If TaskMask bit #11 = 0:
            TaskData = window pointer returned from FindWindow.
            TaskMaster exits and returns wInFrame.
        If the window is not active:
            Call SelectWindow to make the window active.
            TaskMaster exits and returns inNull.
        Else if button down event occurred in a window frame scroll bars:
            TaskMaster does some unorthodox window and port manipulation.
            Calls TrackControl with an action procedure within TaskMaster.
            NOTE: The window owner of frame scroll bar is the Window Manager's.
            The action procedure in TrackMaster performs scrolling and updates.
            TaskMaster exits and returns inNull.
        Else:
            TaskMaster exits and returns wInFrame.

    Else:
        TaskData = window pointer returned from FindWindow.
        TaskMaster exits and returns value returned from FindWindow.

Else:
    TaskMaster exits and returns event code.
```

Window Sizing and Positioning

MoveWindow

inputs: newX - new x coordinate of content region's upper left corner (global).
newY - new y coordinate of content region's upper left corner (global).
theWindow:LONG - pointer to window's port.

output: None.

MoveWindow moves theWindow to another part of the screen, without affecting its size. The top left corner of the window's portRect is moved to the screen point newY,newX. The local coordinates of the window's top left corner remain the same. If the front parameter is TRUE and theWindow, **MoveWindow** makes theWindow the active window by calling **SelectWindow**.

SizeWindow

inputs: newWidth:WORD - new width of window.
newHeight:WORD - new height of window.
theWindow:LONG - pointer to window's port.

output: None.

SizeWindow enlarges or shrinks the portRect of theWindow's grafPort to the width and height specified by newWidth and newHeight, or does nothing if newWidth and newHeight are zero. The window's position on the screen does not change. The new window frame is drawn; if the width of a document window changes, the title is again centered in the title bar, or is truncated if it no longer fits.

ZoomWindow

input: theWindow:LONG - pointer to window's port.

output: None.

ZoomWindow will flip the size and position of theWindow between its current size and position, to its maximum size, passed to NewWindow. If called again, before theWindow is moved or resized, theWindow will be resize and positioned to the size and position before the last **ZoomWindow** was performed. When a **SizeWindow** or **MoveWindow** is performed, while a window is zoomed, the last size becomes the new size and position.

Update Region Maintenance

InvalRect

input: badRect:LONG - pointer to RECT to be added to the update region.

output: None.

InvalRect accumulates the given rectangle into the update region of the window whose grafPort is the current port. This tells the Window Manager that the rectangle has changed and must be updated. The rectangle is given in local coordinates and is clipped to the window's content region.

For example, this procedure is useful when you're calling **SizeWindow** for a document window that contains a size box or scroll bars that are not inside the window's frame. Suppose you're going to call SizeWindow with fUpdate=TRUE. If the window is enlarged, you'll want not only the newly created part of the content region to be updated, but also the two rectangular areas containing the (former) size box and scroll bars; before calling **SizeWindow**, you can call **InvalRect** twice to accumulate those areas into the update region. In case the window is made smaller, you'll want the new size box and scroll bar areas to be updated, and so can similarly call **InvalRect** for those areas after calling **SizeWindow**. As another example, suppose your application scrolls up text in a document window and wants to show new text added at the bottom of the window. You can cause the added text to be redrawn by accumulating that area into the update region with **InvalRect**.

InvalRgn

input: badRgn:LONG - handle of region to be added to the update region.

output: None.

InvalRgn is the same as **InvalRect** but for a region that has changed rather than a rectangle.

ValidRect

input: goodRect:LONG - pointer to a RECT to be removed from the update region.

output: None.

ValidRect removes goodRect from the update region of the window whose grafPort is the current port. This tells the Window Manager that the application has already drawn the rectangle and to cancel any updates accumulated for that area. The rectangle is clipped to the window's content region and is given in local coordinates. Using **ValidRect** results in better performance and less redundant redrawing in the window.

For example, suppose you've called **SizeWindow** with fUpdate=TRUE for a document window that contains a size box or scroll bars not part of the window frame. Depending on the dimensions of the newly sized window, the new size box and scroll bar areas may or may not have been accumulated into the window's update region. After calling **SizeWindow**, you can redraw the size box or scroll bars immediately and then call **ValidRect** for the areas they occupy in case they were in fact accumulated into the update region; this will avoid redundant drawing.

ValidRgn

input: goodRgn:LONG - handle of a region to be subtracted from the update region.

output: None.

ValidRgn is the same as **ValidRect** but for a region that has been drawn rather than a rectangle.

BeginUpdate

input: theWindow:LONG - pointer to window's port.

output: None.

Call **BeginUpdate** when an update event occurs for theWindow. **BeginUpdate** replaces the visRgn of the window's grafPort with the intersection of the visRgn and the update region and then sets the window's update region to an empty region. You would then usually draw the entire content region, though it suffices to draw only the visRgn; in either case, only the parts of the window that require updating will actually be drawn on the screen. Every call to **BeginUpdate** must be balanced by a call to **EndUpdate**. (See "HOW A WINDOW IS DRAWN".) **BeginUpdate** calls can be nested (that is **BeginUpdate** may be called several times, for several different windows, before **EndUpdate** is called for each window).

EndUpdate

input: theWindow:LONG - pointer to window's port.

output: None.

Call **EndUpdate** to restore the normal visRgn of theWindow's grafPort, which was changed by **BeginUpdate** as described above.

Miscellaneous Routines

PinRect

inputs: theRect:RECT - boundary of given point.
theXPt:WORD - the x coordinate of the point to be pinned.
theXPt:WORD - the x coordinate of the point to be pinned.

output: pinnedPt:LONG - point inside theRect nearest to thePt.

PinRect "pins" thePt inside theRect: If thePt is inside theRect, thePt is returned; otherwise, the point associated with the nearest pixel within theRect is returned. (The high-order WORD of the pinnedPt is the vertical coordinate; the low-order WORD is the horizontal coordinate.) More precisely, for theRect (left,top) (right,bottom) and thePt (h,v), PinRect does the following:

- If h < left, it returns left.
- If v < top, it returns top.
- If h > right, it returns right–1.
- If v > bottom, it returns bottom–1.

Note: The 1 is subtracted when thePt is below or to the right of theRect so that a pixel drawn at that point will lie within theRect.

CheckUpdate

input: theEvent:LONG - pointer to an even record.

output: Flag:WORD - TRUE if update event found, else FALSE.

CheckUpdate is called by the Event Manager. From the top to the bottom in the window list, it looks for a visible window that needs updating (that is, whose update region is not empty). If a window with something in its update region is found, an update event for that window is stored in theEevnt and returns TRUE. If it doesn't find such a window, it returns FALSE.

DragRect *(not completed)*

input: theRgn:LONG - handle of region to be dragged.
startX:WORD - starting x coordinate of cursor (global).
startY:WORD - starting y coordinate of cursor (global).
limitRect:LONG - pointer to bounds RECT for dragging.
slopRect:LONG - pointer to RECT which is maximum cursor movement area.
axis:WORD - movement contraint.
actionProc:LONG - address of routine to call while dragging.

output: deltaDrag:LONG - high WORD is vertical delta, low WORD is horizontal delta.

Called when the mouse button is down inside theRgn, **DragGrayRgn** drags a dotted (gray) outline of the region's bounds, which should be in global coordinates, following the movements of the mouse until the button is released. **DragWindow** calls this function before actually moving the window. You can call it yourself to pull around the outline of any region, and then use the information it returns to determine where to move the region. The startY,startX parameters are assumed to be the point where the mouse button was originally pressed, in the global coordinates.

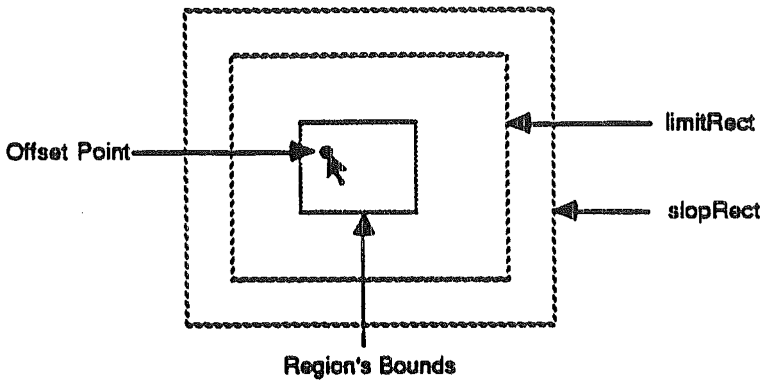
LimitRect and slopRect are also in global coordinates. To explain these parameters, the concept of "offset point" must be introduced: This is initially the point whose vertical and horizontal offsets from the top left corner of the region's enclosing rectangle are the same as those of startY,startX. The offset point follows the mouse location, except that **DragGrayRgn** will never move the offset point outside limitRect; this limits the travel of the region's outline (but not the movements of the mouse). SlopRect, which should completely enclose limitRect, allows the user some "slop" in moving the mouse. DragGrayRgn's behavior while tracking the mouse depends on the location of the mouse with respect to these two rectangles:

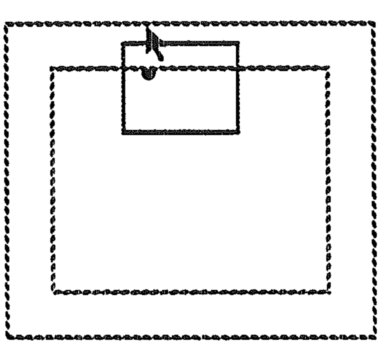
- When the mouse is inside limitRect, the region's outline follows it normally. If the mouse button is released there, the region should be moved to the mouse location.

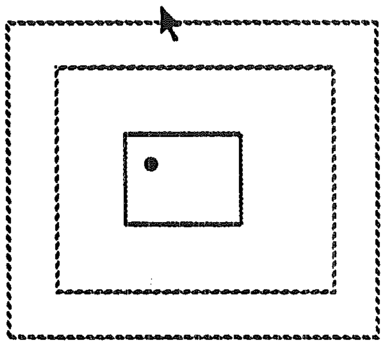
- When the mouse is outside limitRect but inside slopRect, DragGrayRgn "pins" the offset point to the edge of limitRect. If the mouse button is released there, the region should be moved to this pinned location.

- When the mouse is outside slopRect, the outline disappears from the screen, but **DragGrayRgn** continues to follow the mouse; if it moves back into slopRect, the outline reappears. If the mouse button is released outside slopRect, the region should not be moved from its original position.

The diagrams below illustrates what happens when the mouse is moved outside limitRect but inside slopRect, and outside the slopRect.




Outside limitRect, but inside slopRect.


Outside both the limitRect and the slopRect.

The top diagram shows the starting position. As the cursor is moved, an outline of the window is dragged with it. The outline will seem to be glued to the cursor at the offset point. However, if the cursor moves outside of the limitRect, the outline will be left behind, as shown in the lower left diagram. As the cursor is moved outside of the limitRect, but within the slopRect, the outline will get as close to the cursor as possible without letting the offset point leave the limitRect. And finally, if the cursor moves outside the slopRect, the outline will snap back to its starting position. If the cursor moves back into the slopRect, the outline will snap out to get as close as it can.

If the mouse button is released within slopRect, the high-order word of the value returned by DragGrayRgn contains the vertical coordinate of the ending mouse location minus that of startY,startX and the low-order word contains the difference between the horizontal coordinates. If the mouse button is released outside slopRect, both words are zero.

The axis parameter allows you to constrain the region's motion to only one axis. It has one of the following values:

```
CONST  noConstraint   = 0    {no constraint}
       hAxisOnly      = 1    {horizontal axis only}
       vAxisOnly      = 2    {vertical axis only}
```

If an axis constraint is in effect, the outline will follow the mouse's movements along the specified axis only, ignoring motion along the other axis.

The actionProc parameter is a pointer to a procedure that defines some action to be performed repeatedly for as long as the user holds down the mouse button; the procedure should has no parameters. If actionProc is NIL, **DragGrayRgn** simply retains control until the mouse button is released.

Constants

F_HILITED	$0001	Window is highlighted.
F_ZOOMED	$0002	Window is zoomed.
F_ALLOCATED	$0004	Window record was allocated.
F_CTRL_TIE	$0008	Window state tied to controls.
F_INFO	$0010	Window has an information bar.
F_VIS	$0020	Window is visible.
F_MOVE	$0080	Window is movable.
F_ZOOM	$0100	Window is zoomable.
F_GROW	$0400	Window has grow box.
F_BSCROLL	$0800	Window has horizontal scroll bar.
F_RSCROLL	$1000	Window has vertical scroll bar.
F_CLOSE	$4000	Window has a close box.
F_TITLE	$8000	Window has a title bar.
WIND_SIZE	325	Size of WindRec.
wDraw	0	Draw window frame command.
wHit	1	Hit test command.
wCalcRgns	2	Compute regions command.
wNew	3	Initialization command.
wDispose	4	Dispose command.
wNoHit	0	
wInDesk	16	
wInMenuBar	17	
wInSysWindow	18	
wInContent	19	
wInDrag	20	
wInGrow	21	
wInGoAway	22	
wInZoom	23	
wInInfo	24	
wInFrame	27	
BOTTOM_MOST	0	To make window bottom.
TOP_MOST	-1	To make window top.
TO_BOTTOM	-2	To send window to bottom.
noConstraint	0	No constraint on movement.
hAxisOnly	1	Horizontal axis only.
vAxisOnly	2	Vertical axis only.

Data Types

what	Integer	Same as event record.
message	LongInt	Same as event record.
when	LongInt	Same as event record.
where	LongInt	Same as event record.
modifiers	Integer	Same as event record.
TaskData	LongInt	TaskMaster return value.
TaskMask	Integer	TaskMaster feature mask.
wnext	Pointer	Pointer to next window Record.
wport	Port	Window's port.
wstrucRgn	Handle	Region of frame plus content.
wcontRgn	Handle	Content region.
wupdateRgn	Handle	Update region.
wcontrol	Handle	Window's control list.
wFrameCtrl	Handle	Window frame's control list.
wframe	Integer	Bit flags.
FrameColor	Integer	Color of window frame.
TitleColor	Integer	Color of title and bar.
TBarColor	Integer	Color/pattern of title bar.
GrowColor	Integer	Color of grow box.
InfoColor	Integer	Color of information bar.
param_length	Integer	
wFrame	Integer	
wTitle	Pointer	
wRefCon	LongInt	
wZoom	RECT	
wColor	Pointer	
wYOrigin	Integer	
wXOrigin	Integer	
wDataH	Integer	
wDataW	Integer	
wMaxH	Integer	
wMaxW	Integer	
wScrollVer	Integer	
wScrollHor	Integer	
wPageVer	Integer	
wPageHor	Integer	
wInfoRefCon	LongInt	
wFrameDefProc	Pointer	
wInfoDefProc	Pointer	
wContDefProc	Pointer	
wPosition	RECT	
wPlane	LongInt	
wStorage	Pointer	

Error Codes

`ParamLenErr`	1	**NewWindow**	First word of parameter list is the wrong size.
`AllocateErr`	2	**NewWindow**	Unable to allocate window record.
`TaskMaskErr`	3	**TaskMaster**	Bits 12-15 are not clear in TaskMask field of TaskRec.