# WSERVER

```
PSIONICS FILE - WSERVER
=======================
Window server calls
Last modified 1997-04-10
=========================
SPECIAL NOTICE:
This file wouldn't have appeared for much longer if it weren't for Dan Ramage,who sent me enough material to convince me to restart after many months of
neglect.
This file is dedicated to the memory of Felicia Hatunen (see
<http://www.stdc.demon.co.uk/felicia/>). Most of it was written travellingto and from a memorial service for her (in California).
There is a lot of missing and incomplete material in this file. While Psionhave provided me with an SDK and other data, there are a number of gaps inwhat I have, particularly the mapping of registers to C function arguments.Therefore this file contains more than the usual number of @ signs indicatingmissing material. Assistance in filling these gaps would be greatly appreciated (though I don't promise credit).
Processes on the Psion do not have direct access to the screen andkeyboard. Instead, they send messages to a special process called theWindow Server. This process then handles the screen and keyboard,
arbitrating between the various requests - for example, it ensures thatonly the current foreground process gets keystrokes, and handles thecase of different application windows overlapping.
Many calls to the Window Server are buffered; the individual commandsare stored in a buffer, and then sent to the server in a batch. This isdone whenever the buffer is full, by any command requiring a response
from the server, whenever waiting for an event, and by the wFlush call.The calls that return a value but do not flush the command buffer are:
gTextCount, gFontInfo, and wCheckBitmapid.
If an error occurs when a buffered command is executed, all furthercommands are ignored until one that requires a response. This command is
also ignored, and the error is returned instead.
```

Clients of the window server can be in the foreground or background. Onmachines with a small screen, only one client can be in the foreground, andit occupies the whole screen. On machines with a large screen, more than one
client can be in the foreground or be visible at a time.
See the file SYSCALLS.1 for the notation used in this file. In the case offunction $8D, bit 7 of the sub-function number is set to indicate that thefunction obtains a response from the server; this bit is ignored in orderingthe functions in the following list of descriptions. Furthermore, in the
case of Fn $8D Sub $7E and Sub $FF, there are several different functionsdepending on the value of AL.
If a call is marked "fails" then:- if it fails, the resulting behaviour depends on the error handling flag set
  by wDisableLeaves;
- if no new value is shown for AX, it is set to 0 on success.
A common data structure is the "standard rectangle structure". This is defined
here:
  Offset 0 (word): x coordinate of left edge of the rectangle  Offset 2 (word): y coordinate of top edge of the rectangle
  Offset 4 (word): x coordinate of left edge plus width  Offset 6 (word): y coordinate of top edge plus height
Graphics Contexts
---
Many window server calls are described as using a "graphics context". This isa data structure associated with a specific window or bitmap and stored in thewindow server itself. It determines various features about the call, such asthe font used in text operations. At any time there is a "current GC"; calls
labelled "cgc" use that to determine appropriate properties.
A graphics context is specified using a GC description block. This is a blockof memory in the client laid out as follows:
  Offset 0 (byte): graphics mode (0=set, 1=clear, 2=invert)  Offset 1 (byte): text mode (0=set, 1=clear, 2=invert, 4=copy with background)
  Offset 2 (byte): text style
    Bit 0: bold
    Bit 1: underline
    Bit 2: inverse
    Bit 3: double height
    Bit 4: monospaced

```
    Bit 5: italic
    Bit 6: subscript   (v4 only, and only with appropriate font groups)    Bit 7: superscript
(v4 only, and only with appropriate font groups)  Offset 3 (byte): additional graphics options
(v4 only)
    Bits 0 and 1: 0 = black plane, 1 = grey plane, 2 = both planes    Bit 2:        set = double
pixel mode, clear = normal
  Offset 4 (word): font or font groupWherever a font is required, one of the ROM-based fonts
beginning at $4000
can be used. The value $4099 always means the system font.
GC description blocks are normally provided in association with a GC selectormask. This is a
word which is used to determine which values are taken from
the description block, as follows:
  Bit 1: graphics mode
  Bit 2: text mode
  Bit 3: text style
  Bit 4: font
  Bit 5: bits 0 and 1 of additional graphics options (v4 only)  Bit 6: bit 2 of additional
graphics options (v4 only)When creating a new GC, values not taken from the description block
are set
to zero, except the font which is set to the system font.
Events
------
An event is a message sent from the window server to a client to inform it thatsomething has
happened that is not directly related to a function call; forexample, that a key has been
pressed when the client is the foreground client.The event is between 2 and @@ bytes long
depending on its type. Offset 0 is
always a word giving the event type.
In many types of event the words at offsets 2 and 4 have a standard meaning.* "window handle"
(offset 2) - this is the value passed to wCreateWindow. It  is not the same as the window
identifier, and is commonly set to the address
  of a structure including the latter.* "connection handle" (offset 2) - @@@ the value specified
to wConnect
* "tick count" (offset 4) - the window server maintains this value, which it  increments every
tick (1/32 second).
Type 1 - WM_KEY
  Offset  0 (word): type, set to 1
  Offset  2 (word): connection handle
```

```
  Offset  4 (word): tick count
  Offset  6 (word): keycode
  Offset  8 (byte): keyboard modifiers and mouse state
  Offset  9 (byte): unused
  Offset 10 (word): repeat countSent when a key is pressed and the client is the one connected
to the keyboard
(usually because it owns the frontmost foreground window). The keycode isdefined by the key and
modifiers. For example, the 'A' key will generate:
    $0061  alone
    $0041  if shift is pressed or caps lock is on
    $0001  if control is pressed
    $0261  if Psion or alt is pressedThe window server will handle control-plus-number
combinations and generate a
single event. @@@@ special cases
The modifiers are:
  Bit 1: shift key pressed
  Bit 2: control key pressed
  Bit 3: Psion or alt key pressed
  Bit 4: caps lock is on
  Bit 5: num lock is on
  Bit 7: mouse button downThe repeat count is 1 for a single key press. If a key is held down
and is
autorepeating faster than the client accepts new events, the repeat count willbe set to the
number of repeats since the last key event.
Type 2 - WM_MOUSE
  Offset  0 (word): type, set to 2
  Offset  2 (word): window handle
  Offset  4 (word): tick count
  Offset  6 (byte): 1=release, 2=press, 3=motion  Offset  7 (byte): keyboard modifiers and mouse
state
  Offset  8 (word): mouse position x coordinate  Offset 10 (word): mouse position y coordinate
Sent when the mouse button is pressed or released, or the mouse moves andthe appropriate window
has requested motion events. The modifiers are as for
WM_KEY, plus:
  Bit 6: mouse position is outside the visible portion of the windowThis can happen through
grabs and mouse capture.
Type 3 - WM_REDRAW
```

```
  Offset  0 (word): type, set to 3
  Offset  2 (word): window handle
  Offset  4 (word): tick count
  Offset  6 (word): left edge
  Offset  8 (word): top edge
  Offset 10 (word): @@@ right edge or width or what ?  Offset 12 (word): @@@ bottom edge or
height or what ?
Sent when no other events (except WM_USER_MSG) are pending and a window hasan area that needs
redrawing (an "update region"). The rectangle describes some portion (possibly all) of the
update region.
Type 5 - WM_BACKGROUND
  Offset 0 (word): type, set to 5Sent to a client in the foreground when it moves to the
background.
Type 6 - WM_FOREGROUND
  Offset 0 (word): type, set to 6Sent to a client in the background when it moves to the
foreground.
Type 7 - WM_RUBBER
  Offset 0 (word): type, set to 7
  Offset 2 (word): window handle
  Offset 4 (word): tick count
  Offset 6 (word): final status  Offset 8 to  15: standard rectangle structure giving the final
size
The final status can be:
  0 = the final rectangle is the same as the initial one  1 = the rubber band moved but was not
resized
  2 = the rubber band was resized  3 = the rubber band was cancelled; the rectangle is
undefined
  4 = the rubber band was not displayed because the parameters were wrong
Type 8 - WM_USER_MSG
  Offset 0 (word): type, set to 8Sent to a client following a call to wUserMsg if no other
events are pending
(and so can be used to check that condition).
Type 9 - WM_ACTIVE
  Offset 0 (word): type, set to 9
@@@@
Type 11 - WM_CANCELLED
  Offset 0 (word): type, set to 11
```

Sent to a client following a call to wCancelGetEvent.

Type 12 - WM_KEYBOARD_STATE_CHANGE

Sent to the system shell only.

Type 13 - WM_RUBBER_BAND_INIT

  Offset  0 (word): type, set to 13

  Offset  2 (word): window handle

  Offset  4 (word): tick count

  Offset  6 (byte): set to 2

  Offset  7 (byte): keyboard modifiers and mouse state  Offset  8 (word): mouse position x coordinate

  Offset 10 (word): mouse position y coordinateSent to a client on a button press when the window is configured so thatpressing a mouse button starts rubber banding. The window server will suspendall mouse and keyboard processing for all clients until this client calls wRubberBand. If rubber banding is not desired, the cancel flag can be used in the call.

Type 14 - WM_DEICONISE

  Offset 0 (word): type, set to 14

Sent to an iconised client when it moves to the foreground.

Type 15 - WM_ATTACHED

  Offset 0 (word): type, set to 15

  Offset 2 (word): @@@@

  Offset 4 (word): tick count

  Offset 6 (word): attached clientSent to a client when another client has attached itself on top. Note thatthe notifer process attaches itself to a client when that client uses the p_notify@@@ system call.

Type 16 - WM_DETACHED

  Offset 0 (word): type, set to 16

  @@@@

Sent to a client when a previously attached client detaches itself.

Type 17 - WM_COMMAND v3.5

  Offset 0 (word): type, set to 17Sent to a client that is the target of a wSendCommand call, and indicates that wGetCommand should be called.

Type 18 - WM_TASK_KEY Series 3 only

  Offset  0 (word): type, set to 18

  Offset  2 (word): connection handle

  Offset  4 (word): tick count

```
  Offset  6 (word): application key code @@@
  Offset  8 (word): modifiers ? @@@
  Offset 10 (word): repeat count ? @@@
@@@@
Type 19 - WM_TASK_UPDATE v3.5
  Offset 0 (word): type, set to 19Sent to the shell process if any process terminates, whether
or not a client
of the window server.
Type 20 - WM_ON v3.5
  Offset 0 (word): type, set to 20Sent to a client when the machine is switched on and the
client has either
called wInformOnAll, or has called wInformOn and is in the foreground.
Type 21 - WM_ESCAPE
  Offset 0 (word): type, set to 21
  @@@@
Sent on if key events are not selected using wGetEventSpecial. The event issent when the ESC key
is pressed; all pending keystrokes are thrown away.
Type 22 - WM_DATE_CHANGED v4
  Offset 0 (word): type, set to 22Sent to any client which is not in Series 3t compatibility
mode and is in the
foreground, whenever either the date changes, or the machine is switched onon a different date
to when it was switched off.
System calls
------------
Fn $8D Sub $00
wEndRedraw
Ends redrawing, as started by wBeginRedrawWin or related calls.
Fn $8D Sub $01
wEraseTextCursor
Makes the text cursor invisible.
Fn $8D Sub $02
wReleaseMouse
Cancels any call to wCaptureMouse.
Fn $8D Sub $03
gFreeTempGC
Destroys the temporary graphics context, and makes the remembered currentgraphics context be
current again.
```

```
Fn $8D Sub $04
wCancel
@@@
Fn $8D Sub $85
wDisconnect
@@@
Fn $8D Sub $06
wDetachClient
Detaches the client from any process it is attached to (see wAttachToClient)and moves it to the
back.
Fn $8D Sub $07
wCleanUp
Returns the window server to a standard state. This frees any temporarygraphics context, ends
any redraws taking place, and invalidates any
window that has an attached graphics context.
Fn $8D Sub $08
wAttachToForegroundClientAttaches the client to the current foreground client, if different. See
wAttachToClient for details.
Fn $8D Sub $09
wUserMsg
Instructs the window server to send the client a WM_USER_MSG event when thereare no other events
to deliver. Only one such event can be pending for a
client.
Fn $8D Sub $8A
wStartCompute
Inform the window server that the client is about to start intensivecomputation, and its
priority should be set to 112 (background) even ifit is in the foreground. Ignored if the client
has not selected server-
controlled priority handling (see wSetPriorityControl).
Fn $8D Sub $8B
wEndCompute
Cancel the effect of wStartCompute; if the client is in the foreground, sets its priority back
to 128 (foreground). Ignored if the client has
not selected server-controlled priority handling.
Fn $8D Sub $8C
wClientInfo fails
    AX: -> information about the process
```

```
    BX: process to be checked
Returns information about a process:
  Bit 0: set if the process is connected the window server  Bit 1: returns the setting specified
at connection time
  Bit 2: the client is system modal
  Bit 7: server-controlled priority handing is activeIf the process is not a client of the
window server, the call may return zero
or may fail.
Fn $8D Sub $0D
wCloseWindowTree
    BX: window
Destroys the window and all its descendants.
Fn $8D Sub $8E
wInquireWindow
    BX: window
    SI: address of 14 byte information block (see wCreateWindow)The first 10 bytes of the block
are set to the flags, position, and size ofthe specified window. The last 4 bytes of the block
are set to unspecified
values.
Fn $8D Sub $0F
wCaptureMouse
    BX: window
Captures the mouse within the window and its descendants. It will cancel anyactive capture in
another window.
Fn $8D Sub $10
wBeginRedrawWin
    BX: windowValidate the window for redrawing. Validation causes all pixels to be set or
cleared if the background mode is 1 or 2 (this applies to black and greyseparately), and all
future drawing is clipped to the update region (whichis set to the entire window in this case).
Normal drawing resumes when
wEndRedraw is called.
Fn $8D Sub $11
wFree
    BX: identifier
Destroy an object and free all the related resources. The identifier can be
any of:
  * bitmap file handle
```

```
  * bitmap identifier (will also destroy any associated graphics context)  * bitmap sequence
identifier
  * clock identifier
  * font identifier
  * graphics context
  * mouse icon identifier
  * sprite identifier
Fn $8D Sub $12
wMakeInvisible
    BX: window
The window is marked as invisible; a window will not appear on the screen ifit or any of its
ancestors is invisible.
Fn $8D Sub $93
wAttachtoClient fails
    BX: process to attach toAttaches the current process to the specified one. This will cause
its
windows to stay just in front of those of the client it is attached to.Applies only on large
screen servers.
Fn $8D Sub $14
wInitialiseWindowTree
    BX: window
The specified window and its descendants are initialized. This may only bedone once for any
given window, and must be done before it is first drawn on.
Fn $8D Sub $15
wInvalidateWin
    BX: window
Add the entire window to its own update region. This will cause a redrawevent, which can be used
to trigger redrawing of the window.
Fn $8D Sub $16
wValidateWin
    BX: window
Validate the window (causing all pixels to be set or cleared if the backgroundmode is 1 or 2 -
this applies to black and grey separately), and the update
region is deleted.
Fn $8D Sub $17
wMakeVisible
    BX: window
```

Cancels wMakeInvisible on the window.

Fn $8D Sub $18

wClientIconised

    BX: action (0 = deiconise, 1 = iconise)Iconises or deiconises the current process. Iconisation only works on large

screen systems.

Fn $8D Sub $19

wClientPosition

    BX: position

    CX: process IDMove the specified process (zero means the current process) to the given position in the window server task list. Position 0 is the front of thelist; any position greater than or equal to the number of processes attachedto the window server puts the process at the very back of the list.

Fn $8D Sub $9A

wInquireWindowOffset fails

    BX: window A (0 = screen)

    CX: window B

    SI: address of a 4 byte bufferThe buffer is filled in with the position of window B relative to window A:

  Offset 0 (word): x offset

  Offset 2 (word): y offset

Fn $8D Sub $1B

wWindowPosition

    BX: window

    CX: position (0 = front)Moves the window to the specified position within the sibling list (the list

of windows with the same parent).

Fn $8D Sub $1C

wScrollRect

    BX: window

    CX: address of a standard rectangle structure

    DX: address of offset structureThe pixels defined by the rectangle are copied to a new rectangle offset by

the indicated distance. The offset structure is:

  Offset 0 (word): x offset of new rectangle

  Offset 2 (word): y offset of new rectangleCopying does not go outside the window, and does not appear in areas obscured

by other windows or in "invalid" areas waiting for update.

Fn $8D Sub $1D

wRubberBand

BX: window to be sent the WM_RUBBER event CX: window limiting the rubber band (0 = whole screen)

DX: address of an information blockStarts rubber banding, with the rubber band limited to the specified windowor allowed to roam over the screen. When the user completes the rubber banding,

a WM_RUBBER event is sent to the window given. The information block is: Offset 0 to 7: standard rectangle structure giving the initial rectangle Offset 8 to 15: standard rectangle structure giving the inner bounds Offset 16 to 23: standard rectangle structure giving the outer bounds

Offset 24 (word): flags

Offset 26 (word): minimum width

Offset 28 (word): minimum height

Offset 30 (word): maximum width

Offset 32 (word): maximum height

Offset 34 (word): x grid snap value

Offset 36 (word): y grid snap value

@@@@@@@@

A window can be set so that pressing a mouse button starts rubber banding. Inthis case, the press event is sent as a WM_RUBBER_BAND_INIT event instead. Theclient must call wRubberBand (use the @@@@@@ flag to cancel rubber banding if

necessary).

Fn $8D Sub $1E

gCopyRect cgc

BX: standard rectangle structure

CX: address of point definition block

DX: mode (see gFillPattern)Copies a rectangular block from one part of a bitmap to another (the current

graphics context should refer to a bitmap). The top left corner of therectangle is copied to the point given by the definition block:

Offset 0 (word): x

Offset 2 (word): y

Fn $8D Sub $9F

gPeekBit

BX: bitmap identifier, window identifier, or zero (see gSaveBit) CX: address of point

definition block

    DX: number of pixels

    SI: address of bufferCopies a horizontal row of pixels into the buffer. The point definition block

gives the first pixel to be copied:

  Offset 0 (word): x

  Offset 2 (word): y

The pixels are packed 8 per byte of the buffer, in LSB-first order ?@@@@@@In version 4, the grey plane can be used by setting bit 15 of BX.

Fn $8D Sub $20

gDrawPolyLine cgc

    BX: x coordinate of start point

    CX: y coordinate of start point

    DX: address of poly-line control blockDraws a sequence of lines according to the control block, which has the

format:

  Offset     0 (word): number of line segments (N)  Offsets    2 to   5: line segment 1 control block

  Offsets    6 to   9: line segment 2 control block

  ...

where the block is 4N+2 bytes long. The control block for segment M has the

format:

  Offset 4M-2 (word): 2 times x offset, plus 0 for draw or 1 for don't draw  Offset 4M   (word): y offset

Each control block causes a line to be drawn, or not drawn, from the previousend point (or the start point, for the first control block) to a new pointspecified as a relative offset. In version 2 of the window server there is a

limit of 61 control blocks.

Fn $8D Sub $21

gFillPattern cgc

    BX: standard rectangle structure

    CX: bitmap ($4000 = chequered 1 and 0 pixels)

    DX: mode

Fills the specified rectangle with copies of the bitmap. The mode can be:    0 = set each pixel where the bitmap has a 1 pixel

    1 = clear each pixel where the bitmap has a 1 pixel    2 = invert each pixel where the bitmap has a 1 pixel

4 = copy the bitmap to the rectangleIn version 3 and above, CX can also be a window with black plane redrawmethod 3, indicating that the backup bitmap of that window is to be used. In version 4, if both planes have redraw method 3, the appropriate plane is usedfor each selected plane in the target window or bitmap.

Fn $8D Sub $22

gCreateTempGC

BX: window or bitmap

CX: GC selector mask

DX: GC description blockCreates a new temporary graphics context associated with the window or bitmap,

and makes it the current GC. The previous current GC is remembered. While atemporary graphics context exists, the following calls are forbidden:

gCreateGC

gCreateTempGC

gSetGC (unless modifying the temporary graphics context)

wBeginRedrawGC

wBeginRedrawWinGC

wFree for any graphics context

Fn $8D Sub $A3

gCreateGC fails

AX: -> graphics context identifier

BX: window or bitmap

CX: GC selector mask

DX: GC description blockCreates a new graphics context associated with the window or bitmap, and makes

it the current GC.

Fn $8D Sub $24

gSetGC

BX: graphics context (for v3 and above, 0=current GC)

CX: GC selector mask

DX: GC description blockMakes the specified graphics context current, and modifies those properties

specified in the selector mask.

Fn $8D Sub $25

wSetWindow

BX: window to change

CX: flags

    DX: address of a block of information about the windowChanges various properties of a window. The flags are as follows:  Bits 2 to 6, 8, and 10: set the indicated property to the corresponding bit

    of the flags within the information block

  Bit 12: if set, move and resize the window

  Bit 13: if set, change the pointer symbol

  Bit 14: if set, change the background methodNew values are taken from the information block, which is as for wCreateWindow.

Fn $8D Sub $26

wBeginRedrawWinGC

    BX: window

    CX: @@@@

    DX: GC selector mask

    DI: address of a GC description blockValidate the window for redrawing (see wBeginRedrawWin) but using a temporary

graphics context which will be destroyed by wEndRedraw.

Fn $8D Sub $27

gDrawLine cgc

    BX: x coordinate of first end

    CX: y coordinate of first end

    DX: x coordinate of second end

    DI: y coordinate of second endDraws a line between the two locations. If horizontal or vertical, the line

includes the end with the lower coordinates. If diagonal, the line includesall pixels intersected by the diagonal of a rectangle including the top andleft edges but not the bottom and right edges. Note that, in all cases, theorder of the two ends does not affect the line drawn.

Fn $8D Sub $28

gPrintText cgc

    BX: x coordinate of starting point

    CX: y coordinate of baseline

    DX: (text) string

    DI: length of string (maximum 246)Prints the string at the indicated location. Characters not found in the font

will be treated as if they were the highest character code.

Fn $8D Sub $29

gCopyBit cgc

BX: address of point definition block

CX: bitmap identifier or window identifier

DX: standard rectangle structure

DI: mode (see gFillPattern)Copies a rectangular block from a bitmap or a backed-up window (see gSaveBit).The top left corner of the rectangle is copied to the point given by the definition block:

Offset 0 (word): x

Offset 2 (word): yThe function cannot copy from a bitmap to itself. If the destination is in grey plane mode, the grey plane of the source is used if there is one. If thedestination is in both planes mode, a single plane source is copied to bothplanes, and a double plane source has each plane copied to the corresponding
plane.

Fn $8D Sub $AA

wCreateWindow fails

AX: -> new window identifier

BX: parent window (0 = parent is the entire screen)

CX: flags

DX: address of a block of information about the window DI: handle for the new window (must not be zero)

Creates a new window which is a child of the specified window, and gives itthe indicated id and handle (note that the two are not the same, and the formeris used in other calls except where explicitly stated otherwise). The windowcannot be drawn to until wInitialiseWindowTree has initialised it. The flags
are:

Bits 0 to 11: if set, use the corresponding bit from the flags word of the information block; if clear, bits 1, 2, 7, 8, and 9 are cleared, the remainder are inherited from the parent window. Bit 12: set = use coordinates and size, clear = covers entire parent

Bit 13: set = use pointer symbol in information block, clear = use parent's Bit 14: set = new background method, clear = same method as parent

The information block contains the following:

Offset 0 (word): flags

Bit 0: this window is input-only (and therefore invisible) Bit 1: pressing a mouse button starts rubber-banding (see wRubberBand) Bit 2: ignore the mouse when it is within this window

Bit 3: send mouse events when the mouse button is up Bit 4: send mouse events when the mouse button is down

Bit 5: grab the mouse when its button is pressed Bit 6: draw in double pixel mode (v4 only)

Bit 7: visible only when client is in foreground (large screen only) Bit 8: send redraw events to windows with this bit set bfore those with

it clear

Bit 9: do not send redraw events Bit 10: activate the window if it receives a mouse click

Bit 11: rubber banding will terminate when the mouse button is released Offset 2 (word): x coordinate within parent

Offset 4 (word): y coordinate within parent

Offset 6 (word): width

Offset 8 (word): height

Offset 10 (word): identifier of the pointer symbol, where relevant Offset 12 (byte): background redraw method Bits 0 and 1: black plane method (0 = do nothing, 1 = set pixels,

2 = clear pixels, 3 = keep an off-screen bitmap) Bit 3: if set, attempts to draw on the black plane are ignored (v4 only) Bits 4 and 5: grey plane method (0 = do nothing, 1 = set pixels,

2 = clear pixels, 3 = keep an off-screen bitmap) Bit 7: if clear, attempts to draw on the grey plane are ignoredThe choice of keeping an off-screen bitmap cannot be changed after the windowhas been created, while the other options can be changed using wSetWindow.

Fn $8D Sub $2B

wBeginRedrawGC

BX: window

CX: address of a standard rectangle structure

DX: graphics context fields to be set DI: address of a graphics context information block Validate the window for redrawing (see wBeginRedrawWin) with the update regionset to the indicated rectangle and using a temporary graphics context whichwill be destroyed by wEndRedraw (see wBeginRedrawWinGC).

Fn $8D Sub $AC

gPrintClipText cgc

AX: -> number of characters printed

BX: x coordinate of starting point

CX: y coordinate of baseline

DX: (text) string

DI: length of string (maximum 244)

SI: maximum number of pixelsPrints the string at the indicated location as gPrintText, but only enough
characters are printed to fit within the specified number of pixels.
Fn $8D Sub $2D
gPrintBoxText cgc
BX: address of a standard rectangle structure SI: distance from top of rectangle to baseline
AX: alignment (1=right, 2=left, 4=centre)
CX: margin
DX: (text) string
DI: length of string (maximum 236)Clears the specified rectangle, then prints the string within it using textmode 0 (set pixels). The text is clipped to fit within the rectangle. The string is printed in the part of the rectangle excluding a margin area, whoseposition depends on the alignment:
left or right: CX must be >= 0, margin is on the same side, width is CX centre and CX >= 0: margin is on the left, width is CX
centre and CX < 0: margin is on the right, width is -CX
Fn $8D Sub $AE
gOpenBit fails
AX: -> bitmap identifier
BX: (cstr) filename
CX: bitmap number (0 = first or only)
DX: flags
SI: address of an information block
Loads a specified bitmap from a file. The flags are: Bit 0: set for a writeable bitmap, clear for a shared read-only bitmap
Bit 1: place the bitmap in a separate memory segment Bit 2: write the segment name in the information block
Bit 3: must be clear
The information block is written to as follows: Offset 0 (word): set to the width of the bitmap
Offset 2 (word): set to the height of the bitmap Offset 4 to 17: set to the segment name (cstr), only if flag bit 2 is set
Fn $8D Sub $AF
gOpenMouseIcon fails
AX: -> mouse icon identifier
BX: (cstr) file name

```
    CX: number of icon within file
Loads a specified mouse icon from a file.
Fn $8D Sub $B0
gOpenFont fails
    AX: -> font identifier
    BX: (cstr) filename
Opens the specified font file.
Fn $8D Sub $31
gDrawBox cgc
    BX: address of a standard rectangle structureDraws a box of the appropriate size. Note that
a box with width and height of
3 will have exactly one empty pixel inside it.
Fn $8D Sub $B2
wLoadDYL fails
    AX: -> dyl identifier
    BX: (cstr) DYL segment nameLoad a DYL into the window server. The DYL must already have been
loaded into
memory with @@@@@@@
Fn $8D Sub $B3
gCreateBit fails
    AX: -> bitmap identifier
    @@: flags
    @@: address of an information blockCreates an uninitialized writeable bitmap. The flags
are:
  Bit 0: must be clear
  Bit 1: place the bitmap in a separate memory segment  Bit 2: write the segment name in the
information block
  Bit 3: make the bitmap zero-sizedIf the bitmap is zero-sized, nothing can be written to it;
the size can be
increased later. The information block is:
  Offset 0 (word): width
  Offset 2 (word): height  Offset 4 to  17: set to the segment name (cstr), only if flag bit 2
is set
Fn $8D Sub $34
wScrollWin
    BX: window or bitmap
    CX: address of offset structureIdentical to wScrollRect with the rectangle structure set to
```

(0, 0, width,
height).
Fn $8D Sub $35
wDrawTextCursor
    BX: window
    CX: address of text cursor informationAs wTextCursor, but the properties byte is ignored and treated as zero.
Fn $8D Sub $36
wInvalidateRect
    BX: window
    CX: address of a standard rectangle structureAdd the rectangle to the update region of the window. This will generate a
redraw event if necessary.
Fn $8D Sub $37
wBeginRedraw
    BX: window
    CX: address of a standard rectangle structureValidate the window for redrawing (see wBeginRedrawWin) with the update region
set to the rectangle.
Fn $8D Sub $38
wValidateRect
    BX: window
    CX: address of a standard rectangle structureValidate the indicated rectangle of the window (causing all pixels to be setor cleared if the background mode is 1 or 2 - this applies to black and grey
separately), and remove that rectangle from the update region of the window.
Fn $8D Sub $B9
gSaveBit fails
    BX: (cstr) filename
    CX: bitmap identifier, window identifier, or zeroWrites a bitmap to the named file. A window identifier must be for a windowwith a background redraw method of 3, in which case its off-screen bitmap is
used. Zero for the bitmap causes the entire screen to be saved. If the windowoff-screen bitmap, or the screen, has a grey plane then a double-plane bitmap
is written.
Fn $8D Sub $3A
gClrRect cgc

    BX: standard rectangle structure

    CX: 0=set, 1=clear, 2=invert

Sets, clears, or inverts all pixels in the specified rectangle.

Fn $8D Sub $3B

wCallDYL

    BX: dyl identifier

    CX: function number

    DX: number of bytes of data to be passed

    DI: address of data to be passedCall a function in a DYL loaded into the window server that has no result.

Fn $8D Sub $BC

wCallDYLreply fails

    AX: -> result

    BX: dyl identifier

    CX: function number

    DX: number of bytes of data to be passed

    DI: address of data to be passed

    SI: address of area to store a result inCall a function in a DYL loaded into the window server that has a result. Themain result is returned in AX (a negative value is a failure), and other data
(depending on the function) may be written into the area pointed to by SI.

Fn $8D Sub $BD

wSetWinBitmap fails

    AX: -> sequence identifier

    BX: window

    CX: number of bitmaps (1 to 12)

    DX: address of a sequence of information blocksAttaches an animated sequence of bitmaps to the background of a window. Eachtime the background changes the client will be sent a redraw event unless an
appropriate background mode has been selected. There must be an informationblock for each bitmap in the sequence; the blocks are immediately adjacent
in memory. Each block has the form:

  Offset  0 (word): bitmap identifier  Offset  2 (word): x coordinate of location in the window to display bitmap

  Offset  4 (word): y coordinate of location in the window to display bitmap  Offsets 6 to  13: standard rectangle structure giving the part of the bitmap
                    to be displayed  Offset 14 (byte): graphics mode used to copy

bitmap

(0=set, 1=clear, 2=invert, 4=copy with background)  Offset 15 (byte): plane to write to (0=white, 128=grey) (v4 only)  Offset 16 (long): delay in 0.1 seconds before next bitmap is shown

Fn $8D Sub $3E

wChangeWinBitmap

BX: bitmap sequence identifier

CX: position of bitmap (0 to 11)

DX: address of an information blockReplace the appropriate bitmap in a sequence (see wSetWinBitmap) with the new

information in the block.

Fn $8D Sub $BF

wEscapeon

@@@

Fn $8D Sub $40

@@@ RConnect

Fn $8D Sub $C1

wEscapeoff

@@@

Fn $8D Sub $C2

wGetWindowPosition v3 @@@@ ???? [ was = GetPosition] fails

AX: -> position

BX: window

Returns the position of the window in its sibling list (the windows with thesame parent). Has the side effects of wCheckPoint.

Fn $8D Sub $C3

gSaveRect fails

BX: (cstr) filename

CX: bitmap identifier, window identifier, or zero

DX: standard rectangle structureSaves a rectangle from a bitmap, backed-up window, or the screen (see

gSaveBit for details).

Fn $8D Sub $44

wReassignRootWindow v3

BX: window (0 = screen)All future calls within this client will treat the specified window as it it

were the whole screen (e.g. this window will be the parent for those whoseparent is specified as

0).

Fn $8D Sub $C5

wCaptureKey v3 fails

    BX: keycode

    CX: modifier state

    DX: modifier maskCaptures certain key combinations; from now on, these key presses will be sent
to this client whether or not it is in the foreground. The key combinationscaptured are those that will generate the specified keycode, and for whichthose modifiers given in DX have the value given in CX (all other bits inCX must be clear). For example, to only depend on the settings of shift and
control:

| shift | control | CX | DX |
|---|---|---|---|
| up | up | 0 | 6 |
| up | down | 4 | 6 |
| up | either | 0 | 2 |
| down | up | 2 | 6 |
| down | down | 6 | 6 |
| down | either | 2 | 2 |
| either | up | 0 | 4 |
| either | down | 4 | 4 either either 0 0 (if CX is not 0, nothing is captured) |

However, it should be borne in mind that most modifier combinations affectthe keycode as well. The call fails if the same combination is alreadycaptured; if two different combinations capture the same key, the client that
made the earlier request receives the event.

Fn $8D Sub $C6

wCancelCaptureKey v3 fails

    BX: keycode

    CX: modifier state

    DX: modifier maskCancels a call to wCaptureKey with identical arguments. The call fails if the
client has not captured this combination.

Fn $8D Sub $47

wSystemModal v3

    BX: position in task list (0 = front)Makes the current client system modal and sets its position in the task list.Task switching only applies to clients that are in front of the

frontmost
system modal task.
Fn $8D Sub $48
wCancelSystemModal v3
    BX: position in task list (0 = front)Cancels a call to wSystemModal and re-positions the client.
Fn $8D Sub $49
gBorder v3 cgc
    BX: border typeDraws a border just inside the window or bitmap of the current graphics context. The border type is:
  Bit  0: if set, draw the border 1 pixel from the edge  Bit  1: set for 4 pixel corner, clear for 1 or 2 pixel corner
  Bits 2 to 4: allowance for shadow
    0 = no shadow
    1 = remove (but leave space for) single width shadow    2 = remove (but leave space for) double width shadow
    5 = draw single width shadow
    6 = draw double width shadow
  Bit  5: set for special top-of-menu border  Bit  6: set for 1 pixel corner, clear for 2 or 4 pixel corner
  Bit  7: remove any arrow at top right, and repair the border  Bit  8: remove any arrow at bottom right, and repair the border
  Bit  9: include an arrow at top right
  Bit 10: include an arrow at bottom right
Fn $8D Sub $4A
gBorderRect v3 cgc
    BX: standard rectangle structure
    CX: border typeDraws a border just inside the indicated rectangle. The border type is as for gBorder.
Fn $8D Sub $4B
gXPrintText v3 cgc
    BX: x coordinate of starting point
    CX: y coordinate of baseline
    DX: (text) string
    DI: length of string (maximum 244)
    SI: embellishmentsPrints the text, as gPrintText in mode 4 (copy with background), but with embellishments:

```
     0 = none
     1 = inverted
     2 = inverted except the four corners
     4 = underlined
     8 = inverted and no descenders     9 = inverted except the four corners, and no
descenders
    12 = underlined and no descendersEmbellishments with "no descenders" stop on the row below
the baseline, on the
assumption that no characters in the string descend below that point.
Fn $8D Sub $4C
gInvObloid v3 cgc
    BX: address of information blockInverts all the pixels, except the four corner pixels, of
the rectangle
specified by the block:
  Offset 0 (word): left edge
  Offset 2 (word): top edge
  Offset 4 (word): width
  Offset 6 (word): height
Fn $8D Sub $4D
wEnablePauseKey v3Allows the program to be paused by the user by pressing CTRL-S; this is
initially turned on. When paused, the window server will not process anyrequests from the client
(they will be held until the pause is released
by pressing any key).
Fn $8D Sub $CE
wDisablePauseKey v3
Disallow the program from being paused by the use of CTRL-S.
Fn $8D Sub $4F
wsEnable
Turns on the permanent status window. This lies behind any other windows ofthis client and so,
to be seen, all overlaying windows must be resized or
moved.
Fn $8D Sub $50
wsDisable
Turns off the permanent status window.
Fn $8D Sub $51
wInformOn v3.5Instructs the window server to send WM_ON events when the client is in the
foreground.
```

Fn $8D Sub $52

wsUpdate

    BX: reason (1 = name, 2 = time)Updates the contents of the status window to reflect a change elsewhere.

A reason of "name" means that the process name has changed; a reason of"time" indicates that the clock format has changed.

Fn $8D Sub $D3

wsCreateClock fails

    AX: -> clock identifier

    BX: window

    CX: flags

    DX: x coordinate

    SI: difference in minutes between clock and system time @@@

    DI: y coordinateCreates a clock. See wsCreateClock2, which includes all the functionality of this call together with additional features.

Fn $8D Sub $54

wsSetClock

    BX: clock identifier

    CX: difference in minutes between clock and system timeChanges the difference between clock and system time for a clock.

Fn $8D Sub $D5

wSetBusyMsg

   eBX: (cstr) text to be displayed

    CX: location and delay:

      Bits 0-5: delay in half seconds before displaying

      Bit  6:   0 = top, 1 = bottom

      Bit  7:   0 = left, 1 = rightWaits for the required time (during which another call can cancel the

previous request) and then displays the text, flashing, in the indicatedcorner; the text remains until replaced or cancelled (by an emptystring). Equivalent to the BUSY keyword. The text is limited to 20 characters.

Fn $8D Sub $56

wsDisableTemp

Disable the display of a temporary status window via the PSION-MENU keypress.This is a system-wide setting and is initially on.

Fn $8D Sub $57

wsEnableTemp

Enable the display of a temporary status window via the PSION-MENU keypress.This is a system-wide setting and is initially on.

Fn $8D Sub $D8

wSystem

    BX: settings to be changed

    CX: new values for settingsChanges settings within the window server (thus affecting all clients). For

each bit that is set in BX, the property is set to the corresponding bit inCX. The meanings of the bits are:

  Bit @: do not restart the shell (SYS$SHLL) when the window server terminates  Bit @: do not restart the notifier (SYS$NTFY) when the window server

         terminates (v4 only)

  Bit @: attempt to become the notifier process [H4]  Bit @: if the window server is also the notifier, do not report processes

         which terminate abnormally  Bit @: send task update events to the shell when any process terminates [H4]  Bit @: test battery and display low battery warnings if necessary each time

         the machine is switched on [H4]  Bit @: report clients that are failing to respond to events [H4]

  Bit @: disable low battery indicator in the status window [v4]  Bit @: disable SSD indicators in the status window [v4]

  Bit @: disable link indicator in the status window [v4]  Bit @: disable caps lock indicator in the status window [v4]Those bits marked [H4] apply only to HC machines and version 4 of the server;

Series 3t machines will act as if the bit is always set. Those marked [v4]apply only in version 4. The initial values of all settings are taken from

the $WS_FL environment variable.

Fn $8D Sub $D9

wGetProcessList v3.5

    SI: address of a 44 byte bufferThe buffer is filled with a list of all the processes connected to the windowserver; the list is in front-to-back order, and is terminated by a zero word.

Fn $8D Sub $DA

wSendCommand v3.5 fails

    BX: process

    CX: address of the first byte of the data

    DX: number of bytes of data (1 to 127)Sends command data to the specified process. The

process will receive a
WM_COMMAND event, and should then call wGetCommand.
Fn $8D Sub $DB
wGetCommand v3.5 fails
    SI: address of a 127 byte bufferReads the last command data sent by wSendCommand into the buffer. Only one
pending command is held.
Fn $8D Sub $5C
wTextCursor
    BX: window
    CX: address of text cursor informationMake the text cursor visible if it is not, and move it to the window and
location given. The information is:
  Offset 0 (word): x coordinate of baseline
  Offset 2 (word): y coordinate of left edge
  Offset 4 (byte): height of cursor
  Offset 5 (byte): distance from baseline to top of cursor
  Offset 6 (byte): width
  Offset 7 (byte): properties (v4 only)
    Bit 0: clear=rectangle, set=rounded
    Bit 1: clear=flashing, set=steady
    Bit 2: clear=black, set=greyThe distance from baseline to cursor top is 2's complement signed. That is,0 means the top is the baseline, 1 means it is the pixel above, 255 means the pixel below, and so on.
Fn $8D Sub $DD
wAppKeyHandler
@@@
Fn $8D Sub $DE
wInfoMsgCorner
   eBX: (cstr) text to be displayed
    CX: location:
      Bit 6: 0 = top, 1 = bottom
      Bit 7: 0 = left, 1 = rightDisplays the text in the indicated corner for 2 to 2.5 seconds; an empty
string clears any existing text. Equivalent to the GIPRINT keyword. The textis limited to 64 characters.
gSetOpenAddress

Fn $8D Sub $5F

BX: 0=cancel, 1=direct, 2=indirect long, 3=indirect word CX: file offset low word ??? it's declared as ULONG

DX: file offset high word ??? it's declared as ULONG

Modifies the next call to any of:

gInitBit

gOpenBit

gOpenFont

gOpenFontIndex

gOpenMouseIcon

to change the place within the file that reading starts; this is used wherethe font, bitmap, or icon file is embedded in another file. The meaning of BX

is:

0: equivalent to BX=1, CX=0, DX=0

1: DX/CX is the starting position within the file 2: DX/CX is the position within the file of a long giving the start point 3: DX/CX is the position within the file of a word giving the start point

Fn $8D Sub $60

wDrawButton cgc

BX: standard rectangle structure

CX: (cstr) string in button (maximum length 240) DX: button state (0 = normal, 1 = pressed)

Draws a button containing a string.

Fn $8D Sub $E1

wSetTaskKey v3.5 fails

BX: keycode

CX: modifier state

DX: modifier maskCauses the relevant keypresses (see wCaptureKey) to be treated as the "task key". The task key causes the current foreground task to be moved to the backof the list. There can be any number of task keys. On the HC and MC, the TASKkey is preassigned as a task key; on the Series 3 SHIFT-SYSTEM is assigned by

the shell process.

Fn $8D Sub $E2

wSetBackTaskKey v3.5 fails

BX: keycode

CX: modifiers

DX: modifier maskIdentical to wSetTaskKey, except that it sets a "reverse task key" which

brings
the rearmost process to the foreground. On the Series 3 SHIFT-PSION-SYSTEM ispreassigned as the reverse task key.
Fn $8D Sub $E3
wCancelTaskKey v3.5 fails
    BX: keycode
    CX: modifiers
    DX: modifier mask
Cancels a previous call to wSetTaskKey with the same arguments.
Fn $8D Sub $E4
wCancelBackTaskKey v3.5 fails
    BX: keycode
    CX: modifiers
    DX: modifier mask
Cancels a previous call to wSetBackTaskKey with the same arguments.
Fn $8D Sub $E5
@@@ SwExt
Fn $8D Sub $E6
gOpenFontIndex v4 fails
    AX: -> font identifier
    @@: (cstr) filename
    @@: font number within file (0 = first or only font)Opens the specified font from a multiple-font file.
Fn $8D Sub $E7
gInitBit v4 fails
    AX: -> bitmap file handle
    @@: (cstr) filename
    @@: address of a wordOpens a file containing one or more bitmaps, ready for gGetBit and gDrawBit.
The word pointed to by @@ is set to the number of bitmaps in the file. Usingthis facility is more efficient than using gOpenBit several times.
Fn $8D Sub $E8
gGetBit v4 fails
    @@: bitmap file handle
    @@: bitmap number (0 = first or only)
    @@: flags
    @@: address of an information blockLoads a bitmap from the file given by the handle (which

must have been
returned by gInitBit). The other parameters are as for gOpenBit.@@@@ Manual says returns 0.
gOpenBit returns a bitmap identifier !
Fn $8D Sub $E9
gDrawBit v4 fails cgc
    @@: address of point definition block
    @@: bitmap file handle
    @@: standard rectangle structure
    @@: mode (see gFillPattern)
    @@: bitmap number (0 = first or only)Copies a rectangle from a bitmap in a file to the
window or bitmap given bythe current graphics context. Effectively does gGetBit, gCopyBit, and
then
wFree on the bitmap, only much more efficiently.
Fn $8D Sub $EA
gQueryBit v4 fails
    @@: bitmap file handle
    @@: bitmap number (0 = first or only)
    @@: address of 4 byte buffer
The buffer is filled with the size of the specified bitmap:
  Offset 0: width
  Offset 2: height
Fn $8D Sub $6B
wsStatusWindow v4
    BX: window format (0 = off, 1 = small, 2 = large, 3 = Series 3t)Sets the format of the
status window.
Fn $8D Sub $6C
wInformOnAll v4
    @@: new state (0 = don't send, 1 = send)Instructs the window server to send or not send
WM_ON events.
Fn $8D Sub $ED
wsCreateClock2 v4 fails
    AX: -> clock identifier
    BX: address of clock information    CX: (cstr) clock format for type 5 clocks, must be 0 for
all others
Creates a clock, which will then tick automatically. The clock informationhas the following
form:
  Offset  0 (word): window

```
  Offset  2 (word): clock type
    0 = Series 3t small digital
    1 = Series 3t medium variable (36 x 32 pixels)    2 = Series 3t large analogue (66 x 60
pixels)
    3 = Series 3a medium variable (58 x 51 pixels)    4 = Series 3a large analogue (99 x 99
pixels)
    5 = Formatted digital (v4 only)
    6 = Series 3c/Siena medium variable
    7 = Series 3c large analogue
  Offset  4 (word): x coordinate of position
  Offset  6 (word): y coordinate of position  Offset  8 (word): difference in minutes between
clock and system time
  Offset 10 (word): flags [apply only to the types in brackets]    Bit  4: display the date [0,
1, 3]
    Bit  5: display seconds [0, 2, 3 (analogue), 4]
    Bit  6: force analogue [1, 3]
    Bit  7: force digital [1, 3]    Bit  8: add AM/PM indicator if system is in 12 hour mode [0,
1]
    Bit  9: centre text within space added when bit 8 set [0]    Bit 10: add AM/PM indicator if
system is in 12 hour mode [0, 1 (digital)]
    Bit 10: @@@ CLOCK_WITH_DATE_EXTRA
    Bit 11: use grey plane [0, 1, 2, 5?]
    Bit 12: @@@ DBL_PLANE
    Bit 13: right align [@@@]
    Bit 14: @@@ SUMMER_TIME
    Bit 15: enclose in a box [5]
  Offset 12 (word): font (type 5 only)
  Offset 14 (word): style (type 5 only)
    Bit 0: bold
    Bit 1: underline
    Bit 2: inverse
    Bit 3: double height
    Bit 4: monospacedClock types described as "variable" can be either digital or analogue
according
to the system settings or additional bits. For type 5 clocks, the format string
@@@
Fn $8D Sub $EE
```

wSetPriorityControl v4

    BX: new state (0 = off, 1 = on)Sets server-controlled priority handling. When on, the server will change thepriority of a process to 112 when it is in the background or has called wStartCompute, and 128 when in the foreground. The client must not set itsown priority when this is on.

Fn $8D Sub $EF

gConfigureFonts v4 fails

    AX: -> font group identifier

    @@: number of fonts

    @@: address of a sequence of information blocksCreates a font group, which is a set of fonts intended for use in variousstyles (for example, roman, bold, italic, and bold-italic versions of a
basic font would be combined to form a font group). There must be aninformation block for each font in the group; the blocks are immediately
adjacent in memory. Each block has the form:

  Offset 0 (word): font

  Offset 2 (word): style of this font

  Offset 4 (word): style to be applied to this font

  Offset 6 (word): vertical adjustmentWhen using a font group and a style for output, the following process is used.The "best" block is set to the first block. The list of blocks is then scanned
in order; whenever a block is found where all the styles in offset 2 are partof the required style *and* include all the styles in offset 2 of the bestblock, that block becomes the best block. When all blocks have been scanned,the font in offset 0 of the best block is selected, the styles in offset 2 areremoved from those to be used, and then the styles in offset 4 are added. The
resulting styles are applied to the font, and the text is moved vertically bythe distance given in offset 6.

Fn $8D Sub $F0

wInquireStatusWindow v4    AX: -> current format (0 = off, 1 = small, 2 = large, 3 = Series 3t)

    BX: format of interest (as for AX, or -1 for current format)    SI: address of 8 byte buffer The buffer is filled in with the coordinates the status window wouldhave if it had the indicated format; the current format is also returned.

The contents of the buffer are:

  Offset 0 (word): x coordinate of left edge

  Offset 2 (word): y coordinate of top edge

```
  Offset 4 (word): width
  Offset 6 (word): height(When the window is off, it has zero width but full height, at the right
hand edge of the screen.)
Fn $8D Sub $F1
wInquireCompatibility
    AX: -> compatibility flagReturns the state of the Compatibility mode. See wCompatibilityMode
for
details.
Fn $8D Sub $F2
gReadFontHeader v4 fails
    AX: -> length of header actually read (up to 128)
    @@: (cstr) filename
    @@: font number within file (0 = first or only font)
    @@: address of 128 byte buffer
The font header of the specified font is read into the buffer.
Fn $8D Sub $F3
gReadFontGroupHeader v4 fails
    AX: -> length of header actually read (up to 126)
    @@: (cstr) filename
    @@: address of 128 byte bufferThe file must be a multiple font file. The number of fonts is
placed in the
word at offset 0 of the buffer, and the font group header is read into the
rest of the buffer.
Fn $8D Sub $F4
wSetSystemFont v4 fails
    @@: system font to be set
    @@: font, font group, or internal font identifier    @@: text style (internal fonts only)
Changes the font in use for one of the four standard system fonts.    0 = Series 3a system
font
    1 = Series 3t compatibility system font
    2 = Series 3a internal font
    3 = Series 3t compatibility internal font
Fn $8D Sub $F5
wSetSprite v4 fails
    @@: sprite identifier    @@: address of a position structure, or 0 to leave it in the same
place
    @@: number of bitmap set to change (0 = first set in the sequence)    @@: address of an
```

information block, or 0 if all sets remain unchangedMoves the sprite, or change a bitmap set, or both. The position structure andinformation block, if provided, have the same format as for wCreateSprite.

Fn $8D Sub $F6

wCreateSprite v4 fails

AX: -> sprite identifier

@@: window

@@: address of a position structure @@: child window behaviour (0=sprite over windows, 1=sprite under windows)

@@: number of bitmap sets in the sprite @@: address of a sequence of information blocks

Attaches an animated sequence of bitmap sets to a window as a sprite; thesprite will appear in front of everything else in the window. Only one windowof a client may have a sprite at any time. The position structure gives the

initial position of the sprite:

Offset 0 (word): x coordinate

Offset 2 (word): y coordinateThere must be an information block for each bitmap set in the sequence; the

blocks are immediately adjacent in memory. Each block has the form: Offset 0 (word): bitmap for black pixels to be set

Offset 2 (word): bitmap for black pixels to be cleared Offset 4 (word): bitmap for black pixels to be inverted

Offset 6 (word): bitmap for grey pixels to be set Offset 8 (word): bitmap for grey pixels to be cleared

Offset 10 (word): bitmap for grey pixels to be inverted Offset 12 (word): x coordinate of bitmap position relative to the sprite Offset 14 (word): y coordinate of bitmap position relative to the sprite Offset 16 (long): delay in 0.1 seconds before next bitmap set is shown

Any of the bitmaps can be 0 to indicate that there is no such bitmap. Allnon-zero bitmaps must be the same size.

Fn $8D Sub $F7

wsSetList v4 fails

BX: number of strings in the new list (or -1 to show an icon)

eCX: an array of cstrs DX: position in list to be selected (0 = first string, -1 = none)

Sets the "mode", or "diamond" list in the status window. CX should holdthe address of an array of words, each of which holds the address of thestart of a cstr. If BX is -1, the list is replaced by the application

icon; in this case CX and DX are ignored.

Fn $8D Sub $F8

wsSelectList v4

    BX: position in list to be selected (0 = first string, -1 = none)Sets the position of the diamond in the status window.

Fn $8D Sub $79

gShadowText v4 cgc

    @@: x coordinate of starting point

    @@: y coordinate of baseline

    @@: address of information block

    @@: (text) string

    @@: length of string (maximum 234)Prints the text, ignoring the text mode, but applying shadow and lighting

effects according to the information block:  Offset  0 (byte): body colour   ) 0 = black,     1 = white

  Offset  1 (byte): shadow colour ) 2 = grey,      3 = none  Offset  2 (byte): light colour  ) 4 = dark grey, 5 = light grey

  Offset  4 (word): link to shadow (0 = disconnected, 1 = connected)  Offset  6 (word): shadow effect width

  Offset  8 (word): shadow effect height

  Offset 10 (word): lighting effect width

  Offset 12 (word): lighting effect height  Offset 14 (word): additional horizontal gap between characters

Fn $8D Sub $7A

gDrawObject v4 cgc

    @@: object type

    @@: standard rectangle structure

    @@: flags

Draws a special object just inside the specified rectangle. Available object types are:

    0 = 3-dimensional box

and the available flags are:

  Bit 1: set for 4 pixel corner, clear for 1 or 2 pixel corner

  Bit 3: double thickness

  Bit 6: set for 1 pixel corner, clear for 2 or 4 pixel corner

Fn $8D Sub $7B

gBorder2 v4 cgc

    @@: 0 = as gBorder, 1 = black/white/grey 3-D effect

    @@: border typeDraws a border as gBorder, except that it can also generate a 3-D effect on

screens with grey available. @@@ arrows are not documented as available

Fn $8D Sub $7C

gBorder2Rect v4 cgc

    @@: 0 = as gBorderRect, 1 = black/white/grey 3-D effect    @@: standard rectangle structure

    @@: border typeDraws a border as gBorderRect, except that it can also generate a 3-D effect on screens with grey available. @@@ arrows are not documented as available

Fn $8D Sub $FD

wCompatibilityMode

    BX:  new state (0 = off, 1 = on)    SI:  *the same* pointer to a WSERV_SPEC structure as for

         wConnect @@@@Changes the compatibility mode state. When the mode is off, drawing is done in the usual way. When it is on, drawing is done in double pixel mode onthose machines that support it. The initial state is off.

Fn $8D Sub $7E AL $00

wDrawButton2 v4 cgc

    @@: standard rectangle structure

    @@: (cstr) string in button (maximum length 240)    @@: button type (0 = Series 3t, 1 = Series 3a)

    @@: button state (0 = normal, 1 = pressed, 2 = deeply pressed)Draws a button containing a string. Series 3a buttons require the window tohave a grey plane. Series 3t buttons must not be deeply pressed.

Fn $8D Sub $7E AL $01

@@@ InformInactivity

Fn $8D Sub $7E AL $02

wDisableKeyClick v4

    BX: 0 = enable, 1 = disable

Enables or disables key clicks for the current process.

Fn $8D Sub $FF AL $00

gInitMultiSave v4 fails

    AX: -> multisave file handle

    @@: (cstr) filename

    @@: number of bitmapsCreates a file capable of holding the specified number of bitmaps, for use with

gSaveMultiBit and gSaveMultiRect.

Fn $8D Sub $FF AL $01

gSaveMultiBit v4 fails

```
    @@: multisave file handle
    @@: bitmap identifier, or zero for the whole screenSaves the specified bitmap as the next
bitmap in the multisave file created bygInitMultiSave. If the save fails, all further saves to
the file will also
fail (but the file must still be closed with gEndMultiSave).
Fn $8D Sub $FF AL $02
gSaveMultiRect v4 fails
    @@: multisave file handle
    @@: bitmap identifier, or zero for the whole screen    @@: address of a standard rectangle
structure
Saves a rectangle from the specified bitmap as the next bitmap in the multisavefile, as for
gSaveMultiBit.
Fn $8D Sub $FF AL $03
gEndMultiSave v4 fails
    @@: multisave file handle
Closes the multisave file, making it unavailable for further saves.
Fn $8D Sub $FF AL $04
gInquireChecksum v4
    @@: bitmap identifier, window identifier, or zero
    @@: address of a wordChecksums the specified bitmap and writes the checksum to the word
pointed to
by @@. A window identifier must be for a window with a background redrawmethod of 3, in which
case its off-screen bitmap is used. Zero for the bitmap
causes the entire screen to be checksumed.
Fn $8D Sub $FF AL $05
gCreateFontHeader
@@@@
Fn $8D Sub $FF AL $06
wSupportInfo v4
    @@: address of a 32 byte bufferThe buffer is filled in with information about supported
features:
  Offset 0 (word): flags
    Bit 0: set if grey is available    Bit 1: set if Series 3t compatibility mode is available
The remaining flag bits, and the rest of the buffer, is currently set to 0.
@@@@@@@@@
Fn $8D Sub $@@
wsScreenExt
```

```
    @@: address of 8 byte bufferThe buffer is filled in with the size of the screen *excluding*
the current
status window. The format of the buffer is as for wInquireStatusWindow,which should be used for
preference.
Fn $D6 Sub $00 (wConnect) is used to connect to the window server. AllOPL programs, and all
programs using the console device CON: are
connected automatically, and must not use this call.
Fn $D6 Sub $01
gPlayback
    @@: @@@@
@@@
Fn $D6 Sub $02
gCloseMetafile
    @@: @@@@
@@@
Fn $D6 Sub $03
gRecordToMetafile
    @@: @@@@
    @@: @@@@
    @@: @@@@
@@@
Fn $D6 Sub $04
wFlush
Sends any buffered commands to the server.
Fn $D6 Sub $05
wCloseDown
@@@
Fn $D6 Sub $06
wSelect
    @@: @@@@
@@@
Fn $D6 Sub $07
wPanic
    @@: @@@@
@@@
Fn $D6 Sub $08
wGetEvent
```

    @@: address of a 16 byte buffer@@@@ This is async. How does the sync version work ? Is it vsync ?

This call may be made synchronously or asynchronously. If made synchronously,it waits until the window server sends an event to the process, and then copiesthe event into the buffer. If called asynchronously, it writes the value -46into the word at offset 0 of the buffer and returns; when an event is sent fromthe window server, it is written to the buffer and the I/O semaphore is signalled. Further information about events is given below.

Fn $D6 Sub $09

wDisconnect

Disconnects from the window server and frees all associated resources.All programs are automatically disconnected upon exit, so this call is

generally not used by the programmer.

Fn $D6 Sub $0A

wCheckPoint fails

Sends any buffered commands to the server (as wFlush) and reports anyerrors immediately (see wDisableLeaves).

Fn $D6 Sub $0B

gTextWidth fails

    AX: -> width in pixels

    DX: font or font group

    SI: style adjustments

    DI: (text) string

    CX: length of stringReturns the width in pixels of the specified string when displayed in the specified font and style adjustments. The latter are:

    Bit 0: bold

    Bit 1: underline

    Bit 2: inverse

    Bit 3: double height

    Bit 4: monospaced

    Bit 5: italic

    Bit 8: subscript

    Bit 9: superscript

Fn $D6 Sub $0C

gFontInfo fails

    DX: font or font group

    CX: style adjustments (see gTextWidth)

    DI: address of 32 byte bufferFills the buffer with information about the font modified by

```
the style:
  Offset  0 (word): minimum character code
  Offset  2 (word): maximum character code
  Offset  4 (word): height (affected by style)
  Offset  6 (word): ascent (affected by style)
  Offset  8 (word): descent (affected by style)  Offset 10 (word): width of a digit (affected by
style)
  Offset 12 (word): width of widest common character (affected by style)  Offset 14 (word):
flags:
    Bit 0: contains standard ASCII
    Bit 1: contains IBM code page 850
    Bit 2: looks bold
    Bit 3: looks italic
    Bit 4: has serifs
    Bit 5: monospaced
    Bit 6: has a "hang table" @@@@
  Offset 16 to  31: nameThe widest common character is usually an M or W, and is not necessarily
the
widest character in the font.
Fn $D6 Sub $0D
wDisableLeaves
    AX: -> previous value of the flag
    @@: error handling flagSets the error handling mechanism used by the window server calls. If
the
flag is zero, then an error in a command calls the "leave" operation. If itis non-zero, calls
that can fail return an error number in AX. The initialsetting is zero; OPL programs should
always make it non-zero.
Fn $D6 Sub $0E
gCheckBitmapID v3 fails
    @@: bitmap identifierSucceeds (returning 0) if the bitmap identifier is valid. Fails
otherwise.
Fn $D6 Sub $0F
@@@ Alert
Fn $D6 Sub $10
@@@ AlertCancel
Fn $D6 Sub $11
gTextCount fails
```

AX: -> number of characters

DX: font or font group

SI: style adjustments (see gTextWidth)

DI: (text) string

CX: length of string

@@: address of a word holding the available widthAX is set to the length of the longest initial substring that will fit in thenumber of pixels stored in the word pointed to by @@ when displayed in thespecified font and style adjustments (see gTextWidth). The word is altered to hold the number of pixels that would be left over.

Fn $D6 Sub $12

gGetWidthTable fails

DX: font or font group

SI: style adjustments (see gTextWidth)

DI: address of a bufferFills the buffer with the widths of each character in the font as adjusted.

The first byte of the buffer corresponds to the minimum character code, andthe last byte to the maximum character code (see gFontInfo).

Fn $D6 Sub $13

wGetEventSpecial v4

@@: address of a 16 byte buffer

@@: event type maskStarts looking asynchronously for an event as wGetEvent, but only accepts events of specific types. @@@ what happens to the rest ? @@@ based on the

event type mask:

Bit 0: key and task key events

Bit 1: redraw events

Bit 2: foreground, background, and power on events

Bit 3: mouse and rubber band events

Bit 4: all other events

Bit 15: escape eventsCalling wGetEvent is equivalent to calling this function with a mask of $1F.

Escape events are sent only if key events are not selected; in this case, ifthe ESC key is pressed, all pending keystrokes are thrown away and the client

is sent an escape event.

Fn $D6 Sub $14

wGetEventUpdate v4

@@: event type maskIf there is an asynchronous call to wGetEvent or wGetEventSpecial outstanding,it changes the mask specifying the events being looked for. If no call is

```
outstanding, it has no effect.
```

---

Revision #1

Created Thu, Jan 24, 2019 10:36 AM by Alex

Updated Thu, Jan 24, 2019 10:36 AM by Alex