```
This file contains descriptions of the window utility commands.
1
 2
3
              1.
 4
                       Wopen
                       =====
 5
 6
              This command creates a new window with the status of a terminal and
 7
              executes the command given as argument in it (if no command is
 8
              specified, a shell is executed).
 9
10
               The syntax is:
11
                        wopen [-notbwz] [-c \langle n \rangle] [-r \langle n \rangle] [-h \langle n \rangle] [-w \langle n \rangle] [-x \langle n \rangle]
12
                               [-y \langle n \rangle] [-f \langle c \rangle] [-s \langle n \rangle] [-e \langle n \rangle] [\langle command \rangle]
13
14
              Explanation of the options:
15
                 b - Black window.
16
                 w - White window (this is the default).
17
                 n - No window border.
18
                 o - Single (one) line window border.
19
                 t - Double (two) lines window border (this is the default).
20
                 z - Zoom box shall be present in the border.
21
                 c - Number of character columns in the window (default 80).
22
                 r - Number of character rows in the window (default 24).
23
                 h - height of window in pixels.
24
                 w - width of window in pixels.
25
                 x - x coordinate of the lower left corner of the window (default 24
26
27
                      in portrait mode and 152 in landscape mode).
                 y - y coordinate of the lower left corner of the window (default 344
28
                      in portrait mode and 216 in landscape mode).
29
                 f - The default font to be used (default 'A').
30
                 s - Signal to be used to signal that the window has moved, etc.
31
                      (default 0).
32
                 e - Signal to be sent when the close box is used. If not zero, a
33
                      close (exit) box will be present in the border (default 0).
34
35
36
37
               2.
                        Whead
                        =====
38
39
40
               This command inserts a header in a window.
               The syntax is:
41
42
                        whead [-i] [-t] [(header)]
43
44
               Explanation of the options:
45
                 i - Invert the header.
46
                 t - Invert the top header.
47
48
               If no header is given, the present header will be removed.
49
50
51
               3.
                        Wicon
52
                        ----
53
54
55
               This command sets up an icon in a window.
               The syntax is:
56
57
                        wicon [-prielmqszt] [-x \langle n \rangle] [-y \langle n \rangle] [-w \langle n \rangle] [-h \langle n \rangle]
58
59
                               [(sequence)]
60
               Explanation of the options:
61
```

b - Send icon sequence when left mouse button is pressed (default).

```
r - Send icon sequence when left mouse button is released.
63
                i - Invert the icon when the mouse pointer points to it.
64
                e - Send the icon sequence when we enter the icon area.
65
                1 - Send the icon sequence when we leave the icon area.
66
                m - Remove the icon after the icon sequence has been sent.
67
                q - Only send the icon sequence if there is a pending read request
68
69
                    on the window.
                s - Check if option e or l is fulfilled upon set up.
70
                z - Only send the icon sequence if it is the level zero window.
71
                t - The coordinates and sizes are supposed to be given in character
72
73
                x - The x coordinate of the lower left corner of the icon
74
75
                     (default 0).
                y - The y coordinate of the lower left corner of the icon
76
                    (default 0).
77
                w - The width of the icon (default 100).
78
79
                h - The height of the icon (default 100).
80
              (sequence) is the icon sequence to be sent when the icon is chosen.
81
82
83
84
              4.
                      Rmicons
                       -----
 85
86
              This command removes all icons in a window.
87
 88
              The syntax is:
89
90
                      rmicons
 91
 92
93
              5.
                       Wzoom
 94
                       =====
95
 96
              This command sets up a zoom list for a window.
 97
              The syntax is:
98
99
                      wzoom [(zoomlist)]
100
              (zoomlist) is a string of capital letters indicating the fonts which
101
              the zoom list shall consist of. If no (zoomlist) is specified, any
102
              existing zoomlist is removed.
103
104
105
                       Wfont
106
              6.
107
                       =====
108
              This command changes the default font for a window.
109
              The syntax is:
110
111
                       wfont [-x \langle n \rangle] [-y \langle n \rangle] [\langle font \rangle]
112
113
114
              Explanation of the options:
115
                 x - The x coordinate for the middle visible character (default 1).
116
                y - The y coordinate for the middle visible character (default 1).
              (font) is a single capital letter specifying the new font.
117
              If no (font) is specified, the next font in the zoom list for the
118
              window is used instead.
119
120
121
122
              7.
                       Wtop
123
                       ====
```

This command moves a window to the top level. 125 126 The syntax is: 127 128 wtop 129 130 131 8. Wba === 132 133 This command reads the file specified as argument and uses the data to 134 set up a new background pattern for the window handler. It supposes 135 file descriptor 3 to be the window handler "super" channel. 136 137 The syntax is: 138 139 wbg [-n] [(file)] 140 where the '-n' option shall be used if no error messages shall be 141 displayed. 142 If 'file' is not specified, the standard input is read instead. 143 144 145 9. Wask 146 147 ==== 148 This command reads the file specified as argument and uses the data to 149 set up new mouse substitute keys for the window handler. It supposes 150 file descriptor 3 to be the window handler "super" channel. 151 The syntax is: 152 153 wmsk [-n] [(file)] 154 155 156 where the '-n' option shall be used if no error messages shall be 157 displayed. If 'file' is not specified, the standard input is read instead. 158 159 160 10. 161 Wmp 162 === 163 This command reads the file specified as argument and uses the data to 164 set up a new global mouse pointer for the window handler. It supposes 165 file descriptor 3 to be the window handler "super" channel. 166 The syntax is: 167 168 169 wmp [-n] [(file)] 170 where the '-n' option shall be used if no error messages shall be 171 displayed. 172 If 'file' is not specified, the standard input is read instead. 173 174 175 176 11. Widtp 177 ===== 178 This command reads the file specified as argument and uses the data to 179 set up new initial driver and terminal parameters for the window 180 handler. It supposes file descriptor 3 to be the window handler 181 182 "super" channel. 183 The syntax is: 184

widtp [-n] [(file)]

where the '-n' option shall be used if no error messages shall be displayed.

If 'file' is not specified, the standard input is read instead.

### 12. Wshdis

======

This command is the reverse of the window shell preprocessor. It produces a text file from a file produced by wshpp which can be modified and then processed by wshpp again. The syntax is:

```
wshdis [(infile)] [-o (outfile)]
```

Where 'infile' is the input file (default '.window') and 'outfile' is the output file (default standard output).

## 13. Wpictrd

This command reads a rectangle of the picture memory for a virtual screen or the whole screen and writes an optional parameter header followed by the binary data to the standard output. The parameter header is the wpictblk structure (see the w\_structs.h header file). The syntax is:

wpictrd [-p] [-x 
$$\langle n \rangle$$
] [-y  $\langle n \rangle$ ] [-w  $\langle n \rangle$ ] [-h  $\langle n \rangle$ ] [-c  $\langle n \rangle$ ] [-o  $\langle file \rangle$ ]

Explanation of the options:

- p first output a header parameter.
- x x pixel coordinate of the lower left corner of the rectangle to read (default 0).
- y y pixel coordinate of the lower left corner of the rectangle to read (default 0).
- w Width in pixels of the rectangle (default 100).
- h Height in pixels of the rectangle (default 100).
- c The file descriptor (channel) to read the data through (default 0, i.e. standard input).
- o The name of the output file. If not specified, the output is written to the standard output.

### 14. Wdsize

This command sets up a new default size and location for a window. If no arguments are specified, the current size and location of the window will become the default one.

The syntax is:

wdsize [-t] [-x 
$$\langle n \rangle$$
] [-y  $\langle n \rangle$ ] [-u  $\langle n \rangle$ ] [-v  $\langle n \rangle$ ] [-w  $\langle n \rangle$ ] [-w  $\langle n \rangle$ ]

Explanation of the options:

- t The parameters are given in units of font boxes.
- x The lower left corner of the virtual screen (x coordinate).
- y The lower left corner of the virtual screen (y coordinate).
- u The lower left corner of the window (x coordinate).
- v The lower left corner of the window (y coordinate).

### Jul 29 09:27 1985 win\_cmds.doc Page 5

249	w - Width of the window.
250	h - Height of the window.
251	
252	
253	15. Whelp
254	:::::
255	
256	This command changes the sequence sent when the help box is used.
257	The syntax is:
258	
259	whelp [(sequence)]
260	
261	No sequence will be sent if (sequence) is not given.

Aug 5 19:12 1985 wh\_escapes.doc Page 1 1 1985-07-29 2 ABC1600 WINDOW HANDLER ESCAPE SEQUENCES 3 ------4 5 This documentation briefly describes all the escape sequences 6 implemented in the window handler. There are two types: VT100 and/or Facit Twist compatible sequences and sequences private 8 9 to the ABC1600. The sequences are, if possible, compatible with the ones used 10 in the ABC1600 terminal emulator (the console). 11 12 13 1. VT100 and Facit Twist Compatible Escape Sequences 14 15 16 17 1.1 Cursor Up 18 ======== 19 ESC[(Pn)A 20 21 Moves the text cursor (Pn) lines up. The cursor stops at the top 22 margin. If (Pn) is zero or not present, the cursor is moved one line 23 24 upwards. 25 26 27 1.2 Cursor Down 28 -----29 ESC[(Pn)B 30 31 Moves the text cursor (Pn) lines down. The cursor stops at the 32 bottom margin. If (Pn) is zero or not present, the cursor is moved 33 34 one line down. 35 36 37 1.3 Cursor Forward 38 -----39 40 ESC[(Pn)C 41 Moves the text cursor (Pn) positions to the right. The cursor stops 42 at the right margin. If (Pn) is zero or not present, the cursor is 43 44 moved one position to the right. 45 46 47 1.4 Cursor Backward 48 -----49 50 ESC[(Pn)D 51 Moves the text cursor (Pn) positions to the left. The cursor stops 52 at the left margin. If (Pn) is zero or not present, the cursor is 53 54 moved one position to the left. 55 56 57 1.5 Cursor Position -----58 59

Moves the text cursor to the position specified by the parameters.

ESC[(Pn);(Pn)f

60

61 62 ESC[(Pn);(Pn)H

or

The first parameter specifies the line position and the second the column position. If a parameter is 0 or not specified, the cursor is moved to the first line or column.

## 1.6 Set Top and Bottom Margins

### ESC[(Pn);(Pn)r

 Sets the top and bottom margins for the scrolling region. The first parameter is the line number of the first line in the scrolling region and the second the line number of the bottom line. If no parameters are specified, the scrolling region is set to the entire virtual screen. The minimum size of the scrolling region is two lines. The cursor is placed in the home position.

# 1.7 Erase in Display

### ESC[(Ps)J

Erase some part of or the entire virtual screen according to the parameter.

### Parameter Meaning

Erase from and including the current text cursor position to the end of the scrolling region (default).

 Erase from the start of the scrolling region to and including the current text cursor position.

Erase the whole scrolling region.

This escape sequence does not change the current text cursor position.

# 1.8 Erase in Line

### ESC[(Ps)K

 Erases some part of or the entire line where the text cursor is positioned according to the parameter.

### Parameter Meaning

Erase from and including the current text cursor position to the end of the line (default).

Erase from the start of the line to and including the current text cursor position.

2 Erase the entire line.

 This escape sequence does not change the current text cursor position.

## 1.9 Index

#### ESC D

Moves the text cursor one line downward without changing the column

position. If the cursor is at the bottom margin, a scroll up is 125 performed. 126 127 128 Next Line 129 1.10 ======== 130 131 ESC E 132 133 Moves the text cursor to the first position on the next line downward. 134 If the cursor is at the bottom margin, a scroll up is performed. 135 136 137 1.11 Reverse Index 138 -----139 140 141 ESC M 142 Moves the text cursor one line upward without changing the column 143 position. If the cursor is at the top margin, a scroll down is 144 145 performed. 146 147 148 1.12 Save Cursor 149 ========= 150 151 ESC 7 152 Saves the current text cursor position, graphic cursor position, 153 graphic origin, character attributes, and character font. 154 155 156 157 1.13 Restore Cursor 158 -----159 ESC 8 160 161 Restores all things saved by the Save Cursor sequence to the state 162 when the Save Cursor sequence was last used. If no Save Cursor 163 sequence has been sent to the window the text cursor, graphic cursor, 164 and graphic origin are set to their home positions. 165 166 167 168 1.14 Reset to Initial State 169 \_\_\_\_\_\_ 170 171 ESC c 172 On a VT100 terminal this sequence resets it to its initial state. 173 To simulate this in a window, the following things are performed 174 175 when this sequence is received: - The text cursor is put at its home position. 176 - The graphic cursor is put at its home position. 177 - The text cursor apperance is set to the default. 178 - The Set Mode - Reset Mode flags are set to their default values. 179 - The character attributes are set to their default values. 180 - The top and bottom margin of the scrolling region are set to the 181 182 the top and bottom line of the virtual screen. 183 The graphic origin is set to the lower left corner of the virtual 184 screen. Tab stops are set to the default. 185

The graphic pattern tables are set to their default values.

```
- The current font is set to the default font for the window.
187
               - The whole virtual screen is cleared.
188
               - All the LED's on the keyboard are turned off.
189
190
191
                     Tabulation Backward
192
             1.15
                     ______
193
194
             ESC[Z
195
196
              Moves the text cursor left to the next tab stop. The cursor stops at
197
              the left margin.
198
199
200
201
              1.16
                     Horizontal Tabulation Set
202
                      ______
203
              ESC H
204
205
              Set a horizontal tabulation stop at the current text cursor position.
206
207
208
209
              1.17
                     Tabulation Clear
                      ------
210
211
212
              ESC[(Ps)g
213
              If (Ps) is 0 the horizontal tab stop at the current text cursor
214
              position is cleared (default).
215
              If (Ps) is 3 all horizontal tab stops are cleared.
216
217
218
              1.18
                     Character Attributes
219
220
                      -----
221
222
              ESC[(Ps);(Ps);...;(Ps)m
223
224
              Set or reset character attributes according to the parameter(s):
225
                Parameter
                             Meaning
226
227
                             Attributes off.
228
                    0
                             Bold or increased intensity. On the ABC1600 this has
229
                    1
230
                             the same effect as the reverse character attribute.
231
                    4
                              Underscore.
                             Blink. On the ABC1600 this has the same effect as the
232
                    5
233
                              reverse character attribute.
                    7
234
                             Reverse.
235
236
237
              1.19
                     Device Status Report
                      -----
238
239
240
              ESC[(Ps);(Ps);...;(Ps)n
241
              Request to get a report of the specified status. The status is
242
              determined by the parameter(s).
243
244
245
                Parameter
                             Meaning
246
                              Report the text cursor position. The report sequence
247
                    6
```

is ESC[(Pn);(Pn)R where the first parameter

309 310

```
specifies the line and the second the column.
249
                    ?1
                              Report Portrait/Landscape screen mode. This is
250
                              compatible with the Facit Twist terminal. The report
251
                              sequence is ESC[?Pn for portrait mode and ESC[?Ln
252
                              for landscape mode.
253
254
255
256
              1.20
                      Load LEDs
                      ------
257
258
              ESC[(Ps);(Ps);...;(Ps)q
259
260
              Loads the eight programmable LEDs on the keyboard according to the
261
              parameter(s).
262
263
                Parameter
264
                              Meaning
265
                    0
                              Clear LEDs 1 through 8.
266
                    1
                              Light LED 1.
267
                    2
                              Light LED 2.
268
269
                    3
                              Light LED 3.
270
                    4
                              Light LED 4.
                    5
271
                              Light LED 5.
272
                              Light LED 6.
                    6
273
                    7
                              Light LED 7.
274
                              Light LED 8.
275
276
              The default value of the parameter is 0.
              Note that the status of the keyboard LEDs always reflects the LED
277
              status for the top level window.
278
279
280
281
              1.21
                      Set Mode
282
                       =======
283
284
              ESC[(Ps);(Ps);...;(Ps)h
285
              Sets the modes specified by the parameter(s). The different modes
286
287
              are:
288
289
                Parameter
                              Meaning
290
                              Line feed new line mode. When set causes the LF key
291
                    20
292
                               to imply movement to the first position of the
293
                               following line and causes the RETURN key to send both
294
                               CR and LF.
                    ?5
295
                               Screen mode. When set the window is inverted.
                     ?6
                               Origin mode. When set the home position for the
296
297
                               text cursor is at the upper-left position of the
298
                               scrolling region.
299
                    ?7
                               Auto wrap mode. When set, the text cursor will advance
300
                               to the next line when it reaches the right margin.
301
                    ?32
                               Page mode, i.e. the window does not scroll. This is
302
                               compatible with the Facit Twist terminal.
303
                    ?33
                               Underline cursor. This is compatible with the Facit
304
                               Twist terminal.
                    ?34
                               Blinking cursor. This is compatible with the Facit
305
306
                               Twist terminal.
307
                    ?35
                               Cursor off. This is compatible with the Facit Twist
```

terminal.

```
1.22
                      Reset Mode
311
                      -----
312
313
              ESC[(Ps):(Ps):...;(Ps)]
314
315
              Resets the modes specified by the parameter(s). The different modes
316
317
              are:
318
                Parameter
                              Meaning
319
320
                              Line feed new line mode. When reset causes the LF key
321
                    20
                              to imply only vertical movement of the text cursor and
322
                              the RETURN key to send the single code CR.
323
                              Screen mode. When reset the window is not inverted.
                    ?5
324
                              Origin mode. When reset the text cursor home position
                    ?6
325
                              is at the upper-left position of the virtual screen.
326
                              Auto wrap mode. When reset, the text cursor will
327
                    ?7
328
                              not advance to the next line when it reaches the right
                              margin.
329
                              Scroll mode. This is compatible with the Facit Twist
330
                    ?32
331
                               Reverse block cursor. This is compatible with the
                    ?33
332
333
                              Facit Twist terminal.
                              Non-blinking cursor. This is compatible with the Facit
334
                    ?34
                              Twist terminal.
335
                               Cursor on. This is compatible with the Facit Twist
336
                    ?35
                              terminal.
337
338
339
                      Select Character Set
              1.23
340
                      341
342
              ESC(A
                               ESC)A
343
                        or
344
              ESC(B
                        or
                                ESC)B
345
346
347
348
              ESC(Z
                        or
                                ESC)Z
349
350
              Selects the desired font. When changing between fonts of different
351
              sizes, the fonts will be aligned so that the base lines of the fonts
352
              will be the same.
              Note that when the font is changed for a window, the saving of the
353
              text contents of the window will be lost.
354
355
356
              2.
357
                      ABC1600 Private Escape Sequences
358
                       359
360
                      Draw Line
361
              2.1
362
                       -----
363
364
              ESC: (x); (y); (pno); (cno)d
365
              Draws a line from the current graphic cursor position to \langle x \rangle, \langle y \rangle,
366
              using the pattern specified by (pno). If the colour number (cno) is
367
              '1' a normal line is drawn and if it is '0' or not specified the line
368
              is the inverse of that obtained with the colour number '1'. If (pno)
369
```

is not specified, a continous line is drawn.

The graphic cursor position is updated to  $\langle x \rangle$ , $\langle y \rangle$ .

370 371

434

2.6

Draw Inverted Arc

```
373
                        Draw Inverted Line
374
               2.2
                        ______
375
376
               ESC: (x); (y)i
377
378
               Draws a line from the current graphic cursor position to \langle x \rangle, \langle y \rangle by
379
                inverting the corresponding pixels. The line can be removed by drawing
380
               an inverted line a second time.
381
                The graphic cursor position is updated to \langle x \rangle, \langle y \rangle.
382
383
384
               2.3
                        Move Graphic Cursor
385
                        _____
386
387
                ESC: (x); (y) m
388
389
                Positions the graphic cursor at \langle x \rangle, \langle y \rangle.
390
391
392
                2.4
                        Draw Point
393
394
                         ------
395
396
                ESC: (x); (y); (op); (cno)p
397
398
                Changes or reads the pixel at (x),(y). (op) determines the operation:
399
                  If (op) is 0 or not specified, set the pixel.
400
                  If (op) is 1, clear the pixel.
 401
                  If (op) is 2, complement the pixel.
402
403
                  If (op) is 10, the colour of the pixel at (x),(y) is reported:
404
                                                            (cno) is '1' if the pixel is
405
                        ESC: (x); (y); 11; (cno)p
                                                            set, otherwise '0'.
 406
                                                            The specified pixel is outside
407
                        ESC:\langle x \rangle;\langle y \rangle;11p
                                                            the virtual screen.
 408
409
410
                The graphic cursor position is updated to (x),(y) if (op) is 0, 1, or
411
                Note that (cno) is not used for (op) equal to 0, 1, 2, or 10 and may
412
                be left out.
413
414
415
                2.5
416
                        Draw Arc
                         =======
 417
418
419
                ESC: (x); (y); (len); (pno); (cno)a
 420
421
                Draws a circle arc with the origin at \langle x \rangle, \langle y \rangle from the current graphic
                cursor position counter-clockwise with length (len) using the pattern
 422
                (pno). If (pno) is not specified, a continous arc is drawn.
423
 424
                The length (len) is the number of vertical and horizontal pixel steps,
                i.e. a full circle is drawn when (len) is 8 * circle radius.
425
                If the colour number (cno) is '1', a normal arc is drawn and if it is
 426
                'O' or not specified the arc is the inverse of that obtained with the
 427
                colour number '1'.
 428
                The graphic cursor position is updated to the last drawn pixel in the
 429
 430
                arc.
 431
```

2.7

2.8

435 436

ESC: (x); (y); (len)I

Fill Area

-----

ESC: (x); (y); (pno); (cno)f

437 438

Draws a circle arc, with the origin at  $\langle x \rangle, \langle y \rangle$ , from the current graphic cursor position counter-clockwise with length (len) by 439 inverting the corresponding pixels. 440

441 442 443 The length (len) is the number of vertical and horizontal pixel steps, i.e. a full circle is drawn when (len) is 8 \* circle radius. The graphic cursor position is updated to the last drawn pixel in the arc.

Fills a rectangle with the pattern (pno). If (pno) is not specified,

If the colour number (cno) is '1', a normal fill is done and if it is

'O' or not specified, the rectangle is the inverse of that obtained

The rectangle has one of its corners at  $\langle x \rangle, \langle y \rangle$  and the opposite

444

445

446

447

448

449

450

451

452

453 454

455 456

457

458 459

460 461

462

463 464

465

466 467

468 469

470 471 472

473 474 475

476 477 478

483 484 485

490 491

492 493

494 495 496

"goes around corners". If (pno) is not zero the paint does not "go around corners". If the colour number (cno) is '1' a normal paint is done and if it is

'O' or not specified, the paint is the inverse of that obtained with colour number '1'.

The graphic cursor position is updated to  $\langle x \rangle, \langle y \rangle$ .

Note that since paint works directly with the graphic memory, different results may be obtained if the window being painted is overlapped by another window or not.

### with colour number '1'. The graphic cursor position is updated to $\langle x \rangle, \langle y \rangle$ .

Draw Filled Circle

\_\_\_\_\_\_

all pixels in the rectangle are set.

corner at the current graphic cursor position.

# ESC: (x); (y); (rad); (pno); (cno)c

Draws a filled circle with origin at  $\langle x \rangle$ ,  $\langle y \rangle$  and with radius (rad) using the pattern (pno). If (pno) is not specified, all pixels in the

If the colour number (cno) is '1', a normal fill is done and if it is 'O' or not specified, the circle is the inverse of that obtained with colour number '1'.

Paints an area with the pattern (pno). The area to be painted should

be limited by continous lines (curves) generated by previous line,

 $\langle x \rangle, \langle y \rangle$  specifies the starting point for the paint and should be

the area are supposed to consist of set pixels and vice versa. If (pno) is '0' or not specified, the area is painted completely and

within the area. If the pixel at  $\langle x \rangle$ ,  $\langle y \rangle$  is cleared, the limits of

The graphic cursor position is updated to  $\langle x \rangle$ ,  $\langle y \rangle$ .

dot, circle, fill, paint, etc. operations.

#### 2.9 Paint Area -----

ESC: (x); (y); (pno); (cno)F

558

 $\langle sel \rangle = 1$ 

graphic cursor.

497 498 2.10 Move Area 499 500 ======== 501 ESC:(xsrc);(ysrc);(xdest);(ydest);(width);(height);(op)r 502 503 Moves (actually copies) the rectangular area with lower left corner 504 at (xsrc), (ysrc) to (xdest), (ydest). The area has width (width) and 505 506 height (height). If the operation (op) is '0' or not specified the area is moved (copied) as it is, and if it is '1' the area is complemented. 507 508 The graphic cursor position is not updated. 509 Note that only those areas where both the source and destination areas 510 are visible are moved. 511 512 513 514 2.11 Define Pattern -----515 516 ESC:(pno);(hmask);(vmask);(shift);(op)R 517 518 Redefines the pattern (pno) as specified. The pattern is defined for 519 portrait mode and will be tilted 90 degrees when used in landscape 520 521 mode. (hmask) defines a 16 bit horizontal mask used repeatedly on a scan 522 line during fill or when drawing lines or arcs. 523 (vmask) defines a 16 bit vertical mask where each bit determines the 524 operation on the corresponding scan line. If a bit is set (hmask) is 525 used to fill the scan line, otherwise (op) determines the operation: 526 527 Clear the line, rotate (hmask) the number of bits 528  $\langle op \rangle = 0$ 529 given by (shift). Set the line, rotate (hmask) the number of bits 530  $\langle op \rangle = 1$ given by (shift). 531 Use (hmask) but complemented, rotate (hmask) the 532  $\langle op \rangle = 2$ number of bits given by (shift). 533 Leave line as it is, rotate (hmask) the number of bits 534  $\langle op \rangle = 3$ 535 given by (shift). 536  $\langle op \rangle = 4$ Clear the line, no rotate. Set the line, no rotate. 537  $\langle op \rangle = 5$  $\langle op \rangle = 6$ Use (hmask) but complemented, no rotate. 538 Leave line as it is, no rotate. 539  $\langle op \rangle = 7$ 540 541 (pno) can be in the range 1 - 15. Pattern number zero can not be 542 redefined. 543 (shift) can be in the range 0 - 15. Only (hmask) is used by the draw line and draw arc escape sequences. 544 545 546 547 2.12 Set Text Cursor 548 ------549 550 ESC:(sel)H 551 The text cursor is positioned at the position of the graphic cursor 552 553 according to (sel): 554 The upper left corner of the font box is placed at the 555  $\langle sel \rangle = 0$ 556 graphic cursor.

The lower left corner of the font box is placed at the

The left edge of the base line for the font box is 559  $\langle sel \rangle = 2$ placed at the graphic cursor. 560 561 Note that when this escape sequence is sent to a window, the saving of 562 the text contents of the window will be lost. 563 564 565 566 2.13 Mouse Report 567 -----568 569 ESC: (sel)M 570 571 This escape sequence is used to get a report of the current mouse 572 pointer position. The report is, depending on (sel), only sent when the mouse pointer or the mouse buttons have changed. 573 574 575  $\langle sel \rangle = 7$ The report is sent immediately if the mouse has changed since the last report. Otherwise the report 576 577 is sent as soon as the mouse changes. A change is 578 either a mouse movement or a status change of a mouse 579 button. 580 The report sequence is: 581 582 ESC:(x);(y);(buttons)P 583 584 where  $\langle x \rangle$  and  $\langle y \rangle$  is the position of the mouse pointer. If the mouse pointer is outside the virtual 585 screen, the reported position will be at the virtual 586 587 screen border. 588 (buttons) is '1' if the left button is pressed, '2' if the middle button is pressed, '3' if both the left and 589 middle buttons are pressed, and 'O' if no button is 590 591 pressed. 592  $\langle sel \rangle = 8$ Identical to (sel) = 7, except that reports are only 593 sent when the left or middle buttons changes. 594 595 Note that mouse reports are only sent to the top level window. 596 597 598 2.14 Device Status Report 599 ------600 601 ESC:(sel)n 602 603 Reports the status of different devices, determined by (sel): 604 605  $\langle sel \rangle = 1$ Reports the graphic cursor position. The report 606 sequence is: 607 608 ESC: (x); (y)R 609 610 where  $\langle x \rangle, \langle y \rangle$  is the current graphic cursor position.  $\langle sel \rangle = 2$ 611 Reports the mouse position and button status. This is 612 identical to the Mouse Report escape sequence with 613 (sel) = 7 (ESC:7M), except that the report is sent 614 immediately. 615 Note that reports are only sent to the top level 616  $\langle sel \rangle = 3$ 617 Reports the size of the virtual screen and the current 618 font. The report sequence is: 619

ESC: (vsx); (vsy); (fsx); (fsy); (bl); (fno)W

```
621
                                (vsx) and (vsy) are the x and y pixel sizes,
622
                                respectively, of the virtual screen, (fsx) and (fsy)
623
                                are the x and y pixel sizes of the current font box,
624
                                (bl) is the base line for the font box, and (fno) is
625
                                the ASCII code for the name of the current font.
626
627
628
               2.15
                       Set Graphic Origin
629
                        ______
630
631
632
               ESC:\langle x \rangle;\langle y \rangle 0
633
               Sets the graphic origin to \langle x \rangle, \langle y \rangle. The graphic cursor position is
634
635
               All coordinates given by the graphic escape sequences are relative
636
637
               to the graphic origin.
               Note that the mouse position is always reported relative to the lower
638
               left corner of the virtual screen.
639
640
641
               2.16
                        Clear All
642
                        ========
643
644
               ESC:J
645
646
               Clear window and home cursors, etc. as follows:
647
648
                 - The text cursor is set to 1,1.
649
                 - The graphic cursor is set to 0,0.
650
                 - The graphic origin is set to 0,0.
651
                 - The scroll region is reset to the whole virtual screen.
652
                    If the current character font is the same as the default font for
653
                     the window, the text contents of the window will be started to be
654
                     remembered again.
655

    The whole virtual screen is cleared.

656
657
 658
659
               2.17
                        Load Key LEDs
                        ------
660
661
               ESC: (sel); (sel);...; (sel)q
662
663
664
               Loads the LEDs on the INS and ALT keys according to the parameter(s).
 665
                 (sel) = 0
                                Clear both the LEDs.
666
                                Light the INS key LED.
                  \langle sel \rangle = 1
 667
                                Light the ALT key LED.
                  \langle sel \rangle = 2
668
 669
670
               If no parameter is specified, the LEDs are cleared.
               Note that the status of the keyboard LEDs always reflects the LED
 671
               status for the top level window.
672
 673
674
675
               2.18
                        Private Set Mode
                        -----
 676
677
678
               ESC: (sel); (sel); ...; (sel)h
 679
680
               Sets the ABC1600 private modes specified by the parameter(s). The
 681
               different modes are:
```

Phased pattern mode. When set, the patterns obtained (sel) = 2683 when using the fill area, draw filled circle, paint 684 area, and spray escape sequences will be phased. 685 686 687 Private Reset Mode 2.19 688 ------689 690 ESC: (sel); (sel);...; (sel)1 691 692 Resets the ABC1600 private modes specified by the parameter(s). The 693 694 different modes are: 695  $\langle sel \rangle = 2$ Non-phased pattern mode. When reset, the patterns 696 obtained when using the fill area, draw circle, paint 697 area, and spray escape sequences will not be phased. 698 699 700 2.20 701 Spray 702 ===== 703 704 ESC: (x); (y); (pno); (op)s 705 This escape sequence manipulates the pixels which are set both in the 706 spray mask and in the pattern specified by (pno), according to the 707 operation (op). 708 (x),(y) is the lower left corner of where to put the 32x32 pixels 709 710 If (pno) is not specified, '0' is used and if (op) is not specified, 711 '0' is used. 712 The following operations can be performed: 713 714 All pixels which are set both in the spray mask and in  $\langle op \rangle = 0$ 715 the pattern are set and the remaining pixels are 716 cleared (replace). 717 All pixels which are set both in the spray mask and in  $\langle op \rangle = 1$ 718 the pattern are set. The remaining pixels are left 719 unaffected (set). 720 All pixels which are set both in the spray mask and in  $\langle op \rangle = 2$ 721 the pattern are cleared. The remaining pixels are left 722 723 unaffected (reset). All pixels which are set both in the spray mask and in 724  $\langle op \rangle = 3$ the pattern are complemented. The remaining pixels are 725 left unaffected (complement). 726 727 728 The spray mask for a window can be altered by a request to the window 729 handler. For most applications of this escape sequence, the window must 730 probably be set to phased pattern mode in order to give a meaningful 731 732

The current graphic cursor position is updated to (x),(y).

1985-07-29 Peter Andersson Luxor Datorer AB

THE WINDOW SHELL PREPROCESSOR - WSHPP

### 1. Introduction

Wshpp is a preprocessor for the window shell - wsh. As input it takes a text file describing the menu's and other things to be used to start programs, open pull down menus, etc. when using the ABC1600 window handler. The output is in a compact binary format which wsh can handle efficiently.

Wshpp can also produce single data structures to be used by other programs when creating windows, setting up icons, etc. By always using wshpp when creating the data to be used to call the window handler, future incompatibility problems can be avoided.

It should be pointed out that the format of the text input file is of a fairly low level, instead it is possible to use most of the facilities of the window handler. If higher level routines is desired (for example the input is just a collection of independent icons), it is recommended that a program is written which as output produces a text file which can be processed by wshpp.

# 2. Command Syntax

The syntax of wshpp is:

wshpp [-n] [(infile)] [-x (struc) (outfile) -x (struc) (outfile)...]
 [-o (outfile)]

(infile) is the input text file. If it is not specified, the standard

input is used instead.
The '-o' option specif

The '-o' option specifies the filename of the wsh data output file. The '-x' option with its two following arguments specifies a single structure to be output to a file (see part 4).

If no '-x' or '-o' options are given, the wsh data is written to the file .window, which is the file wsh reads by default. No wsh data file is generated if no '-o' and one or more '-x' options are specified. All or some of the outfiles may be replaced by a dash (-), in which case the corresponding data is written to the standard output (all messages displayed by wshpp are written to the standard error output).

This is intended to be used together with pipes.

The '-n' option is used if no output file at all shall be generated.

All error messages displayed by wshpp are by default in english.

However if the environment variable LANGUAGE is set to 'swedish', all

error messages are displayed in swedish instead.

### 3. The Format of the Input File

The input file consists of descriptions of data items and action items.

The data items is (the structures referred to are the ones used to communicate with the window handler and can be found in the documentation for the handler):

63	window	Data for a window (the winstruc structure).
64	icon	Data for an icon (the winicon structure).
65	string	A string to be used in both landscape and portrait
66	*	mode.
67	pstring	A string to be used in portrait mode only.
68	lstring	A string to be used in landscape mode only.
69	pointer	Data for the layout of a mouse pointer (the npstruc
70		structure).
71	header	Data for a window header (the headstruc structure).
72	environ	Environment strings used to modify the environment
73		in both portrait and landscape screen mode.
74	penviron	Environment strings to be used in portrait mode only.
75	lenviron	Environment strings to be used in landscape mode only.
76	directory	A directory pathname.
77	command	A command (the file name and the arguments).
78	flags	Window flags data (the flgstruc structure).
79	zoomlist	Zoom list data (the zoomlst structure).
80	substitute	Mouse substitute keys (the substit structure).
81	background	Data for a background pattern (the chbgstruc
82		structure).

### The action items are:

init	Describes what to do on initialization.
menu	Describes a menu window.
choice	Describes a choice which can be made from a menu window.
action	Describes the action when a certain choice has been chosen.
terminal	Describes a terminal window, i.e. a window running
	a program.

# 3.1 Data Items

The description of a data item consists of its name, which is the name of the item, immediately followed by a number. A colon separates the name from the data. The data either consists of

- (i) one string (string, pstring, lstring, directory),
- (ii) several strings separated by commas (environ, penviron, lenviron, command), or
- (iii) keywords (with corresponding values) and flags (window, icon, pointer, header, flags, zoomlst, substitute, background).

The string in (i) is the rest of the line after the first colon. The strings in (ii) are those between the first colon and a comma or a newline, between two commas, or between a comma and a newline. Data items in (iii) consists of 4-letter keywords, optionally followed by a value, separated by colons. If it is a numerical value, the keyword shall be followed by a '\#' character and the numerical value. The numerical value can be a decimal number, an octal number, or a hexadecimal number. The syntax of the different numbers are the same as in the C language: A number starting with a zero is interpreted as an octal number, a number starting with 'Ox' or 'OX' is interpreted as a hexadecimal number, otherwise it is interpreted as a decimal number

If the value is a string, the keyword shall be followed by an '=' character and the string terminated by a colon or a newline.

A flag consists of just a keyword and if it is present the flag is set, otherwise it is reset.

The backslash (\) can be used as an escape character in strings. This works as in the C language (it has been augmented by '\e' which means ESCAPE, 27 decimal).

Leading and trailing spaces and tabs are significant in all strings. A line can be continued on the next line by ending the line with a backslash.

125

126

127

128 129

### 3.1.1 Window

======

134 135 136

137

138

139

140

The window data item gives the data for a window. Every keyword has a corresponding member or flag in the winstruc structure (see the documentation for the window handler). In the following list the corresponding structure member or flag is listed inside paranthesis and a '#' character indicates that it is a numerical value, otherwise it is a flag.

141 142

184

185

186

7.47		
143	Keyword	Description
144		
145	pxor#	<pre>(wp_xorig) The x coordinate in portrait mode of the</pre>
146		lower left corner of the virtual screen.
147	lxor#	(wl_xorig) The x coordinate in landscape mode of the
148		lower left corner of the virtual screen.
149	pyor#	(wp_yorig) The y coordinate in portrait mode of the
150	• •	lower left corner of the virtual screen.
151	lyor#	(wl_yorig) The y coordinate in landscape mode of the
152	·	lower left corner of the virtual screen.
153	pxsi#	(wp_xsize) The horizontal size in portrait mode of the
154	•	virtual screen.
155	lxsi#	(wl_xsize) The horizontal size in landscape mode of
156		the virtual screen.
157	pysi#	(wp_ysize) The vertical size in portrait mode of the
158	, .	virtual screen.
159	lysi#	(wl_ysize) The vertical size in landscape mode of the
160	·	virtual screen.
161	pvxo#	(wp_vxorig) The x coordinate in portrait mode of the
162	·	lower left corner of the window relative to the lower
163		left corner of the virtual screen.
164	lvxo#	(wl_vxorig) The x coordinate in landscape mode of the
165		lower left corner of the window.
166	pvyo#	(wp_vyorig) The y coordinate in portrait mode of the
167		lower left corner of the window.
168	lvyo#	(wl_vyorig) The y coordinate in landscape mode of the
169	·	lower left corner of the window.
170	pvxs#	(wp_vxsize) The horizontal size in portrait mode of
171		the window.
172	lvxs#	(wl_vxsize) The horizontal size in landscape mode of
173		the window.
174	pvys#	(wp_vysize) The vertical size in portrait mode of the
175		window.
176	lvys#	(wl_vysize) The vertical size in landscape mode of the
177		window.
178	colr#	(w_color) Background colour in the window.
179		O = Black, 1 = White.
180	bord#	(w_border) The type of the window border.
181		The different types are (N = No border, S = Single
182		line border, D = Double lines border):
183		
101		

Border Left

type side

Right

side

------

Upper

side

Lower

side

187		0	N	N	N	N		
188		1	S	S	S	S		
189		2	D	D	D	D		
190		3	D	S	S	S		
191		4	S	D	S	\$		
192		5	S	S	D	S		
193		6	S	S	S	D		
194		7	D	D	S	S		
195		8	D	S	D	S		
196		9	D	S	S	D		
197		10	S	D	D	S		
198		11	S	D	S	D		
199		12	\$	S	D	D		
200		13	D	D	D	S		
201		14	D	D	S	D		
202		15	D	S	D	D		
203		16	S	D	D	D		
204	_							
205	pfnt#			initial f		portrait n	mode (ASCI	Ί
206	• •			nt A is 65				
207	lfnt#			initial f				
208	usr <b>b#</b>	_		e maximal				
209				et up in t				
210	tsig#			signal use				
211				top level.			i always s	ets
212				ero for me				l
213	nsig#			signal us				
214				n the top			w suell al	ways
215				to zero f			ndan that	:.
216	rsig#			signal use itself. T				
217				ııseır. m enu window				une
218 219				enu winuow ne 'stxt'			-	ı
220				is signal				•
221	csig#			signal to				
222	colds			ne close b			3363 III U	
223	hscr			scroll le			es shall b	e
224				e border.		23	-	•
225	vscr			scroll up	and do	in boxes s	shall be	
226				e border.				
227	cbox	•		close box	shall b	e present	t in the	
228				indow shel				ıu
229		windows.						
230	sbox	(BX_SIZE	E) The	size box	shall be	e present	in the bo	order.
231	mbox	(BX_MOVE	) The	move box	shall be	e present	in the bo	rder.
232	zbox			zoom box				order.
233	avis	_		oll left/r	-			
234				e whole vi				
235	bbox	_	I) The	blow up b	ox shall	l be prese	ent in the	•
236	t •	border.	.\ -1					
237	hbox			help box				rder.
238	pmod			ait mode c				
239	lmod			ape mode				
240 241	stxt	Screen.	vi/ 94,	ve the tex	r conte	ira ni fu	e virtudi	
241 242	sbmp		TMAD)	Save the b	itman c	nntante a	f the virt	la l
242 243	20⊞h	screen (			ייישה רו	meches 0	. PIIC AT( )	,uu1
243 244	lock		•	e window	on the	ton loval		
245	novr			vindow mus		•		ther
246	11011	window.	*	1143		. oroniupi	- sa by will	21101
247	ncur		R) Tex	t cursor	not visi	ble.		
248	nmov			window mus			r change s	size.
							•	

249	alls	(ALLSCR) The window must be the whole virtual screen.
250	spec	(SPECIAL) Special window.
251	kscr	(KEYSCROLL) Make sure that the text cursor is visible
252		in the window everytime a key is pressed.
253	WSCY	(WRITSCROLL) Make sure that the text cursor is visible
254		in the window everytime something has been written to
255		the window.
256	amsp	(ALTMPNT) Allocate space to store a private mouse
257	•	pointer for the window.
258	rltv	(RELATIVE) Add the window relative to the parent
259		window.
260	ncpi	(NOCPIN) Prevents text from being copied into this
261		window.
262	ncpo	(NOCPOUT) Prevents text from being copied from this
263		window.
264	text	(TXTSIZE) The size of the virtual screen, the window,
265		and the origin of the window are supposed to be given
266		in term of characters instead of pixels.
267	wgrp	(WGROUP) This window shall belong to a window group.
268	rulo	(REL_ULC) This window shall follow its parent window
269		relative the upper left corner of the parent (not
270		meaningful if the window is not a child window).
271	rurc	(REL_URC) This window shall follow its parent window
272		relative the upper right corner of the parent (not
273		meaningful if the window is not a child window).
274	rllc	(REL_LLC) This window shall follow its parent window
275		relative the lower left corner of the parent (not
276		meaningful if the window is not a child window).
277	rlrc	(REL_LRC) This window shall follow its parent window
278		relative the lower right corner of the parent (not
279		meaningful if the window is not a child window).
280		

The following is an example of a small window put somewhere in the middle of the screen (only portrait mode coordinates are given):

```
window5:pxor#300:pyor#500:pxsi#100:pysi#100:pvxo#0:pvyo#0:pvxs#100:\
    :pvys#80:colr#1:bord#2:pfnt#0x41:\
    :pmode:stxt:cbox:sbox:mbox
```

Note that all values which are not specified are guaranteed to be zero.

## 3.1.2 Icon

The icon data item gives the data for an icon. Every keyword has a corresponding member or flag in the winicon structure (see the documetation for the window handler). An '=' character after the keywords means that the value is a string.

	110/110/ 40 1104	no onde ono ratao to a conting.
299		
300	Keyword	Description
301		
302	pxor#	(ip_xorig) The x coordinate in portrait mode of the
303		lower left corner of the icon.
304	lxor#	(il_xorig) The x coordinate in landscape mode of the
305		lower left corner of the icon.
306	pyor#	(ip_yorig) The y coordinate in portrait mode of the
307		lower left corner of the icon.
308	lyor#	(il_yorig) The y coordinate in landscape mode of the
309		lower left corner of the icon.
310	pxsi#	(ip_xsize) The horizontal size in portrait mode of the

311		icon.
312	lxsi#	(il_xsize) The horizontal size in landscape mode of
313		the icon.
314	pysi#	(ip_ysize) The vertical size in portrait mode of the
315		icon.
316	lysi#	(il_ysize) The vertical size in landscape mode of the
317		icon.
318	cseq=	(i_cmdseq[]) Character sequence to be sent by the
319		icon.
320	pmod	(I_PMODE) Portrait mode coordinates given.
321 322	lmod	(I_LMODE) Landscape mode coordinates given.
323	pres rlse	(I_PRESS) Send sequence when left button is pressed. (I_RELEASE) Send sequence when left button is
324	1126	released.
325	inve	(I_INVERT) Invert icon when we are pointing to it.
326	entr	(I_ENTER) Send sequence when we are moving into the
327	CII DI	icon area.
328	leav	(I LEAVE) Send sequence when we are leaving the icon
329	1047	area.
330	rmov	(I_REMOVE) Remove the icon when a sequence has been
331		sent.
332	rqst	(I_RQST) Only send the sequence if there is a pending
333		read request to the window.
334	schk	(I_SETCHK) Check if 'entr' or 'leav' is fulfilled when
335		setting up the icon.
336	lzer	(I_LZERO) The sequence is sent only if the window is
337		at the top level.
338	text	(I_TEXT) The icon coordinates are supposed to be in
339		character units.
340		
341		example puts the icon in the lower left corner of a
342	virtual scree	n (only portrait mode coordinates are given):
343		#0 #0!#00!#F0 1000 \
344		#0:pyor#0:pxsi#80:pysi#50:cseq=\200:\
345 346	:pmuc	l:pres:rlse:inve:rqst
345 347	Note that all	values which are not specified are guaranteed to be
348	zero.	values willow are not specified are guaranteed to be
349	2010.	
350		
351	3.1.3 Strin	g, pstring, and lstring
352		
353		
354	To set up the	string
355	•	
356	I like	
357	WINDOWS!	
358		
359	using string,	pstring, or lstring, looks like:
360		1.11. A - HTHPOHOL
361		like\nWINDOWS!
362		like\nWINDOWS!
363	15tr 1ng19:1	like\nWINDOWS!
364 365		
366	3.1.4 Point	or
367	3.1.4 (01)(	
368		
740	The pointer d	ata itam gives the data for a mouse pointer lavout

The pointer data item gives the data for a mouse pointer layout. Every keyword has a corresponding member in the npstruc structure (see the documentation for the window handler).

```
373
                Keyword
                               Description
374
375
                xsiz#
                               (np_xsize) The with of the mouse pointer.
376
                ysiz#
                               (np_ysize) The height of the mouse pointer.
377
                xpnt#
                               (np_xpnt) Pointing part of the mouse pointer, x
378
                               coordinate.
379
                               (np_ypnt) Pointing part of the mouse pointer, y
                ypnt#
380
                               coordinate.
381
                andm#
                               (np_and[]) A series of 16 AND masks used to construct
382
                               the mouse pointer. The different elements are
383
                              separated by commas.
384
                               (np_or[]) A series of 16 OR masks used to construct
                orma#
385
                              the mouse pointer. The different elements are
386
                               separated by commas.
387
388
              The following is an example of a black hair cross mouse pointer:
389
390
                pointer7:xsiz#31:ysiz#31:xpnt#15:ypnt#15:\
391
                        :andm#Oxfffeffff,Oxfffeffff,Oxfffeffff,Oxfffeffff,\
392
                              Oxfffeffff,Oxfffeffff,Oxfffeffff,Oxfffeffff,\
393
                              Oxfffeffff,Oxfffeffff,Oxfffeffff,Oxfffeffff,\
394
                              Oxfffeffff,Oxfffeffff,Oxfffeffff,Ox00000001,\
395
                              Oxfffeffff,Oxfffeffff,Oxfffeffff,Oxfffeffff,\
396
                              Oxfffeffff,Oxfffeffff,Oxfffeffff,Oxfffeffff,\
397
                              Oxfffeffff,Oxfffeffff,Oxfffeffff,Oxfffeffff,\
398
                              Oxfffeffff,Oxfffeffff,Oxfffeffff,Oxffffffff\
399
                        :orma#0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
400
401
              Note that all values which are not specified are guaranteed to be
402
              zero.
403
404
405
              3.1.5
                      Header
406
407
              The header data item gives the data for a window header. Every
408
409
              keyword has a corresponding member or flag in the headstruc structure
410
              (see the documentation for the window handler).
411
412
                Keyword
                              Description
413
414
                head=
                              (h hdr[]) The header string.
415
                invh
                               (H_INVHD) Invert the window header.
                invt
                              (H_INVT) Invert the top window header.
416
417
418
              The following is an example of the header 'MY PROGRAM':
419
420
                header17:head= MY PROGRAM :invt
421
422
423
              3.1.6
                      Directory
424
425
              To specify the directory pathname /usr/sven/bin, use the line:
426
427
428
                directory4:/usr/sven/bin
429
430
              3.1.7
431
                      Environ, penviron, and lenviron
432
                      -----
433
```

These data items specifies how the environment for the program

496

=======

```
435
               shall be modified before it is executed by wsh. If the specified
436
               environment variable already exist, the old one is replaced. Otherwise
               the environment variable is added to the environment list.
437
438
               To specify PATH to be '/usr/mydir/bin' and TERM to be 'vt100', use
439
               the line:
440
441
                 environ3:PATH=/usr/mydir/bin,TERM=vt100
442
443
              By modifying the environment it is possible to tell programs, which
444
               uses termcap, the size of the virtual screen. If the size of the
445
              virtual screen is 132 columns times 33 lines, use:
446
                 environ7:TERM=win,TERMCAP=w0\win\w:co#132:li#33:tc=abc1600w:
447
448
               'abc1600w' is an entry in the termcap file which should be used for
449
450
               this purpose only.
451
              The syntax for penviron and lenviron is equivalent.
452
453
454
455
              3.1.8
                       Command
456
                       ======
457
              To specify the 'ls -1' command, use the line:
458
459
460
                command1:/bin/ls,ls,-l
461
              '/bin/ls' is the file name, 'ls' is argument 0, '-l' is argument 1.
462
463
464
              3.1.9
                      Flags
465
466
                       =====
467
              The flags data item gives the data for new window flags. Every
468
              keyword has a corresponding flag in the flastruc structure (see the
469
              window handler documentation).
470
471
                               Description
472
                Keyword
473
                               (LOCK) See the description of the window data item.
474
                lock
                               (NOOVER)
475
                 novr
                               (NOCURSOR)
476
                ncur
477
                nmov
                               (NOMOVE)
478
                alls
                               (ALLSCR)
479
                               (KEYSCROLL)
                 kscr
                               (WRITSCROLL)
480
                 WSCY
481
                               (NOCPIN)
                 ncpi
                               (NOCPOUT)
482
                ncpo
483
                 rulc
                               (REL_ULC)
                               (REL_URC)
484
                rurc
485
                rllc
                               (REL_LLC)
                rlrc
                               (REL LRC)
486
487
              The following example can be used to set the LOCK flag for the window
488
              in the example in section 3.1.1:
489
490
                flags56:lock
491
492
493
494
              3.1.10 Zoomlist
```

Jul 29 16:	09 1985 wshpp.doc	Page 9		
497	The zoomlist	data item gives the data for a zoom list. Every keyword		
498		has a corresponding member or flag in the zoomlst structure (see the		
499		documentation for the window handler).		
500				
501	Keyword	Description		
502				
503	plst=	<pre>(zp_list[]) The set of fonts to be used in portrait</pre>		
504		mode.		
505	llst=	<pre>(zl_list[]) The set of fonts to be used in landscape</pre>		
506		mode.		
507	pmod	(Z_PMODE) Portrait mode list given.		
508	lmod	(Z_LMODE) Landscape mode list given.		
509				
510	The following	; is an example of a zoom list (only data for portrait		
511	mode is giver	mode is given) which will make it possible to toggle between the		
512	window's defa	window's default font and the font F:		
513				
514	z00 <b>m</b> ]	list7:plst=F:pmod		
515				
516				
517	3.1.11 Subst	titute		
518		:::::		
519				
520	The substitut	e data item gives the data for a set of mouse substitute		
521	keys. Every l	keyword has a corresponding member in the substit		

keys. Every keyword has a corresponding member in the substit structure (see the window handler documentation).

322	Structure (Sec	e the window handler documentation).
523		
524	Keyword	Description
525		
526	init#	(c_initflg) Flag indicating if the substitute keys are
527		enabled or not after the set up (1 if enabled, 0 if
528		not).
529	onof#	(c_keys[S_ONOFF]) Key used to toggle the substitute
530		keys on or off.
531	mpup#	(c_keys[S_MPU]) Move mouse pointer up.
532	mpdo#	<pre>(c_keys[S_MPD]) Move mouse pointer down.</pre>
533	mple#	<pre>(c_keys[S_MPL]) Move mouse pointer left.</pre>
534	mpri <b>#</b>	<pre>(c_keys[S_MPR]) Move mouse pointer right.</pre>
535	mpul#	<pre>(c_keys[S_MPUL]) Move mouse pointer up - left.</pre>
536	mpur#	<pre>(c_keys[S_MPUR]) Move mouse pointer up - right.</pre>
537	mpdl#	<pre>(c_keys[S_MPDL]) Move mouse pointer down - left.</pre>
538	mpdr#	<pre>(c_keys[S_MPDR]) Move mouse pointer down - right.</pre>
539	lpup#	<pre>(c_keys[S_LMPU]) Move mouse pointer up a long step.</pre>
540	lpdo#	<pre>(c_keys[S_LMPD]) Move mouse pointer down a long step.</pre>
541	lple#	<pre>(c_keys[S_LMPL]) Move mouse pointer left a long step.</pre>
542	lpri <b>#</b>	<pre>(c_keys[S_LMPR]) Move mouse pointer right a long step.</pre>
543	lpul#	<pre>(c_keys[S_LMPUL]) Move mouse pointer up - left a long</pre>
544		step.
545	lpur#	<pre>(c_keys[S_LMPUR]) Move mouse pointer up - right a long</pre>
546		step.
547	lpdl#	(c_keys[S_LMPDL]) Move mouse pointer down - left a
548		long step.
549	lpdr#	(c_keys[S_LMPDR]) Move mouse pointer down - right a
550		long step.
551	pc <b>md</b> #	(c_keys[S_PCMD]) Replacement for the left mouse
552		button.
553	cwin#	(c_keys[S_CHWIN]) Replacemet for the right mouse
554		button.
555	mtxt#	(c_keys[S_MCA]) Replacement for the middle mouse
556		button.
557	st <b>ep#</b>	(c_step) Step length for a normal move of the mouse
558		pointer.

620

terminal

559 lstp# (c 1step) Step length for a long move of the mouse 560 pointer. 561 562 The following is an exemple of the set up of the mouse substitute 563 keys: 564 565 substitute1:init#0:onof#0xfe:mpup#0xa1:mpdo#0xa3:mple#0xac:\ 566 :mpri#0xa4:mpul#0xad:mpur#0xa5:mpdl#0xaf:mpdr#0xa7: 567 :lpup#0xb1:lpdo#0xb3:lple#0xbc:lpri#0xb4:lpul#0xbd:\ 568 :lpur#0xb5:lpdl#0xbf:lpdr#0xb7:pcmd#0xcc:cwin#0xce:\ 569 :mtxt#0xcd:step#4:lstp#10 570 571 572 3.1.12 Background ======== 573 574 575 The background data item gives the data for a background pattern. 576 The keyword has a corresponding member in the chbgstruc structure (see the documentation for the window handler). 577 578 579 Keyword Description 580 581 bmap# (cb bitmap) The bit pattern of a 16 x 16 pixels area 582 representing the pattern. The 16 elements shall be 583 separated by commas. 584 The following is an example of a white background pattern: 585 586 587 background2:bmap#0xffff,0xffff,0xffff,0xffff,\ 588 0xffff,0xffff,0xffff,0xffff,\ Oxffff,Oxffff,Oxffff,Oxffff,\ 589 Oxffff,Oxffff,Oxffff,Oxffff 590 591 592 Note that all values which are not specified are guaranteed to be 593 zero. 594 595 596 3.2 Action Items ------597 598 599 The description of an action item consists of its name, which is the 600 name of the item in most cases followed by a number. A colon separates 601 the name from the description part, which consists of data items, 602 action items, or in some cases some special actions. 603 604 605 3.2.1 Init 606 607 The init action consists of a list of actions to be performed upon 608 609 initialization. They are executed in the specified order. The 610 following things can be specified to be performed on initialization: 611 612 Item Description 613 614 substitute The keys used as substitute for the mouse. No keys 615 will be set up if substitute is not present. 616 background A new background pattern. The default pattern is used 617 if no background is present. 618 pointer The layout of the global mouse pointer. If no pointer

is specified, the default mouse pointer is used.

Open a terminal window with a program running in it.

action

621	menu	The starting menu window. This must be specified.
622	inverse	Set the screen to inverse video. This is a special
623		action and no number shall be given.
624	normal	Set the screen to normal video. This is a special
625		action and no number shall be given.
626		
627		action can be specified and therefore no init number
628	shall be given	
629	An example:	
630		
631	init:substit	utel:menu3
632		
633	7.0.0	
634	3.2.2 Menu	
635	====	
636	The	describes a second data a sull dam second
637		n describes a menu window, a pull down menu, etc.
638	ine rollowing	things can be specified:
639	Item	Description
640	116#	Description
641 6 <b>42</b>	window	Data for the window to be used as menu. If the window
	MILLOOM	
643 644		already is open, wsh checks if the window already
645		contains the desired strings and icons, and if so these are not set up once more. However if the
646		•
646 647		contents is new, the new icons are set up and the new strings are displayed. One and only one window must
648		be specified.
649	header	The header of the menu window. The header is optional.
650	choice	Describes the choices which it is possible to make
651	CHOICE	from this menu. If no action is specified, at least
652		one choice must be specified.
653	action	The specified action will be executed directly without
654	detion	waiting for a choice from the mouse. If any choices
655		have been given, they are ignored.
656	string	Text and graphic contents of the menu window.
657	pstring	Text and graphic contents of the menu window.
658	lstring	Text and graphic contents of the menu window.
659	pointer	The layout of the mouse pointer when it points into
660	,	this menu. If no pointer is specified, the global
661		mouse pointer is used. The 'amsp' flag for the menu
662		window must be set to make it possible to set up a
663		private mouse pointer.
664		
665	An example:	
666		
667	menu5:window	11:choice20:choice21:choice22:pstring13:lstring13
668		
669	If both an act	ion and choices are given, a warning message is issued.
670		
671		
672	3.2.3 Choice	
673	======	
674	The states and	to consider a fore of all also called a large and a confidence of
675		ion connects an icon with the actions to be performed
676 677		is chosen. The following two things must be specified
677 678	in a choice:	
678 679	Item	Description
680	10公田	nesot thermi
681	icon	The icon.
682	action	The actions to be performed when the above icon is

The actions to be performed when the above icon is

742

743

744

pointer

683 chosen. 684 685 An example: 686 687 choice9:icon7:action17 688 689 690 3.2.4 Action 691 ===== 692 693 The action consists of a list of actions to be performed. The actions 694 will be executed in the same order as they are specified. The 695 following items may be specified in the list: 696 697 Ttem. Description 698 699 flags New window flags for the current menu window. Set new mouse substitute keys. 700 substitute 701 background Set up a new background pattern. 702 pointer Set up a new global mouse pointer. 703 terminal Open a terminal window with a program running in it. 704 menu Go to the specified menu. 705 706 The following special actions may be specified in an action list 707 (no number shall be specified after these special actions): 708 709 Special action Description 710 711 close Close the current menu window. restore Restore the screen. 712 713 inverse Set the screen to inverse video. Set the screen to normal video. 714 normal Move the current menu window to the top level. 715 top 716 turn Turn the screen. Log out. This will only work if there are not 717 logout 718 any open terminal windows. Wsh takes care of 719 checking this. 720 At least one 'menu' must be given. If several are given, a warning is 721 issued. A warning also appears if a 'menu' does not end the list (in 722 this case the actions after the 'menu' will never be executed). 723 724 An example: 725 726 action18:flags5:terminal10:close:menu7 727 728 729 3.2.5 Terminal 730 731 732 The terminal action describes a window to be used to run a program. 733 The following can be specified in a terminal description: 734 735 window Data for the window to be used as terminal. At most one may be specified. If no window is specified, the 736 737 command will be executed with '/dev/null' as standard 738 input, output, and error output. 739 header Optional header for the terminal window. 740 zoomlist Optional zoom list for the terminal window.

Layout of the mouse pointer to be used when pointing

into the terminal window. If no pointer is specified,

the global mouse pointer is used instead. The 'amsp' flag for the terminal window must be set to make it

745		possible to set up a private mouse pointer.
746	icon	Optional icons to be set up before the execution
747		of the program starts.
748	string	Optional strings to be written to the terminal window
749		before the execution of the program starts.
750	pstring	As above.
751	lstring	As above.
752	directory	An optional directory to move to before the execution
753		of the program starts. If no directory is specified,
754		the current directory for the program when it starts
755		will be the same as the one where wsh were started
756		from.
757	super	If present, the "super" channel will be open as file
758		descriptor 3 in the program. This is a special action
759		and no number shall be given.
760	wait	Causes with to wait for the command to finish. This is
761		a special action and no number shall be given.
762	environ	Optional modification of the environment.
763	penviron	As above, but only for portrait mode.
764	lenviron	As above, but only for landscape mode.
765	command	Specifies the program to be executed in the terminal
766		window.
767		

### An example:

### 3.3 More about the Format

The number of all the numbered items must be an integer greater than or equal to one.

All lines starting with a '#' character are supposed to be comments and ignored.

All the data and action items may be given in any order.

### 4. Writing Single Structures to File

To output, for example, a single window structure (winstruc) to a file, the '-x' option is used.

Suppose we have a text file - menu.wd - which contains a description of a window named window3. The command

wshpp menu.wd -x window3 win3

will write the window data structure described by window3 to the file win3. All the remaining data in the input file is ignored. The following data items can be extracted and written to a file in this way:

 800
 window

 801
 header

 802
 icon

 803
 pointer

 804
 flags

 805
 zoomlist

 806
 substitute

background

1985-07-07 Peter Andersson Luxor Datorer AB

THE WINDOW SHELL - WSH

## 1. Introduction

The window shell - wsh - is an interface between the user and the ABC1600 window handler. To know what to do, wsh starts by reading a data file. This file is created by the window shell preprocessor - wshpp. The documentation for wshpp covers most of the things concerning wsh, so this documentation just describes the syntax of wsh and gives some notes of how wsh behaves in different situations.

## 2. Command Syntax and Start Up

wsh [-n] [(file)]

The syntax of wsh is:

 (file) is the input data file. If it is not specified, wsh tries to read the file '.window' in the current directory, and if this fails it finally tries to read the file '/etc/.window'.

Normally wsh (after reading the data file) activates the window handler. The '-n' option tells wsh not to do this. In this case wsh assumes that the window handler already has been activated and that the file descriptor for the window handler "super channel" is 3. This can be used together with the 'wait' and 'super' special actions (see the documentation for wshpp) to start "sub-window shells". If wsh is started from another terminal than the console or from a window, the ordinary shell - sh - is executed instead.

Error messages from wsh are by default in english. However if the environment variable LANGUAGE is set to 'swedish', all error messages

## 3. Some Notes of the Behaviour of Wsh

are displayed in swedish instead.

- When wsh are going to get a command from a menu window it first checks if the window is already open (if not wsh opens it). Then it is checked that the contents (header, strings, and icons) is the desired and if not the old header and icons are removed and the new header and icons are set up and the specified strings are written to the menu window.

- Wsh automatically sets up a redraw signal ('rsig') for all menu windows which have not the 'stxt' flag set and takes care of redrawing them when necessary. If the 'stxt' flag is set, wsh sets 'rsig' to 0 and supposes the window handler to take care of the redrawing of the window. Note that because wsh manipulates 'rsig' for menu windows, the same window data description should not be used both for menu and terminal windows.

There is no need to specify the character sequence ('cseq') to be sent by the icon for icons used in menu windows as wsh uses its own sequences. As for window data, the same icon data description

63 should not be used both for menu and terminal icons. 64 - 'tsig' and 'nsig' are always set to 0 and the 'cbox' flag is 65 66 cleared for menu windows. The reason is that wsh can not handle 67 these things. 68 - The cursors are not moved to their home positions and the window 69 70 is not cleared before the specified strings are displayed in a 71 window. These things must, if necessary, be included in the 72 strings. Be especially careful with strings which must be rewritten 73 by wsh to update menu windows. 74 75 Strings are always written in the specified order. 76 77 All terminal windows are set up as controlling terminals, i.e. 78 '/dev/tty' refers to the window. 79 80 - The processes running in different terminal windows belongs to 81 different process groups. 82 - Only file descriptors 0, 1, and 2 (standard input, output, and 83 84 error output) and sometimes 3 (the "super channel") are open when 85 the command specified in a terminal description is executed. 86 87 - When the command in a terminal description is executed, all signals 88 are set to default except those signals specified by 'tsig', 89 'nsig', 'rsig', and 'csig' which are ignored. 90 91 - The current directory for wsh is always the directory where it was 92 started from. Terminals will initially have the same current directory if no 'directory' is specified. 93 94 95 - The command specified in terminals can be shell scripts and wsh 96 automatically searches for the command in all the directories specified by the PATH environment variable. 97 98 99 When handling the 'turn' special action, wsh checks that there are no windows open, except for menu windows. If not, all menu 100 101 windows are closed and the window shell executes the 'init' action 102 in the new screen mode. 103 104 When handling the 'logout' special action, wsh ignores it if there

are any windows open, except for menu windows.

 1985-07-29, Peter Andersson, Luxor Datorer AB

## ABC1600 WINDOW HANDLER

The ABC1600 Window Handler is, as indicated by the name, implemented as a handler under ABCenix and has special calls to open new windows, move windows around, return the status of a window, remove windows, etc. It also automatically takes care of things like:

- Moving and altering sizes of windows, using a mouse.
- Convert pointing to a specified area inside a window to a command sequence (e.g. pointing to icon's).
- Moving text between windows.

### 1. The Model

When several windows are present on the screen each of them is thought of as being at a certain level. The window on the top is at level 0 and it receives all the input from the keyboard. All the other windows are at lower levels; the window one step from the top is at level 1 and so forth.

To switch to another window (i.e. attach the input from the keyboard to another window), that window must be put at level 0. When this is done, all windows previously at higher levels than the new level 0 window are moved one level down. The level 0 window can also be moved to the bottom, making all other windows moving one level up. The output from the processes connected to a certain window are always sent to that window, regardless of if it is at level 0 or not. Each window emulates a DEC VT100 terminal augmented by ABC1600 private escape sequences. The ABC1600 private escape sequences are compatible with or similar to their counterparts in the ABC1600 terminal emulator. See wh\_escapes.doc for further details regarding the escape sequences.

# 2. Starting and Terminating the Window Handler

The window handler is started by giving the command:

/usr/window/whgo

This is a start-up program, usually started by the rc script, which mounts itself on the '/win' directory and waits in the background until the window handler is activated. This is done with an open request, which in C can look like:

fd = open("/win/activate", 2);

The file descriptor returned (greater than or equal to zero if no errors) can later be used to disactivate the handler and also to issue some special requests to it.

On activation of the window handler, 'whgo' performs some initializations and then executes a portrait or landscape mode version of the handler, depending on the direction of the screen. A close request is used to disactivate the window handler:

64 65 66

67 68 69

70

71 72 73

74 75 76

77 78 79

80

81 82 83

85 86 87

84

88 89 90

91

92 93 94

95 96 97

> 98 99

100 101 102

107 108 109

114 115

116 117

118

119 120 121

122 123 124 When the handler receives this request it sends hangup signals to all processes in the windows, resets the screen, and then executes 'whgo'

The terminate signal will terminate the window handler in a controlled manner witout executing 'whgo'.

#### 3. Opening Windows -----

When the window handler has been activated, windows can be opened by issuing an open request to the handler:

```
fd = open("/win", 2);
```

This will not create a window on the screen, it just tells the handler to allocate space for a new window. The returned value - 'fd' - is greater than or equal to zero if the open was successful and is used to write to, read from, send I/O control requests to, and close the window.

To acctually create the window on the screen, the Wincreat request is used (see below).

#### Closing Windows 4. -----

To close a window, a close request shall be sent to the handler with the file descriptor obtained when the window was opened:

```
close(fd);
```

This will cause the handler to remove the window from the screen.

#### 5. Write to and Read from Windows ------

To write to a window the standard write system call can be used with the file descriptor obtained when the window was opened:

```
write(fd, bp, bc);
```

To read from (through) a window, i.e. get input from the keyboard, the read system call can be used:

```
cnt = read(fd, bp, bc);
```

#### 6. Window Requests ------

The following is a description of all the requests which are implemented to manipulate the windows from other processes. They are all macros, and the definitions of them can be found in the file (win/w\_macros.h). The constant definitions can be found in <win/w\_const.h>, the structure declarations in (win/w\_structs.h), and new variable type declarations can be found in (win/w\_types.h). The requests returns a negative value if they fail. The unions included in most of the structures below are reserved for future use. To guarantee compatibility with future versions, the

```
125
               member of the union must be zero.
126
127
128
               6.1
                       Create Window
129
                       -----
130
131
               To create a window the following request is used:
132
133
                       Wincreat(fd, bp);
134
                       int
                                       fd;
135
                       struct winstruc *bp;
136
137
               'fd' is the file descriptor obtained from the open request and the
138
               structure winstruc looks like:
139
140
                       typedef
                                        short pix_d;
141
                       typedef
                                        short cur_d;
142
                       typedef
                                        char
                                               sint;
143
                       typedef unsigned short word;
144
                       typedef unsigned long uflags;
145
146
                      struct winstruc
147
                       {
148
                              pix_d
                                       wp_xorig;
149
                               pix_d
                                       wl_xorig;
150
                              pix_d
                                       wp_yorig;
151
                               pix_d
                                       wl_yorig;
152
                              pix_d
                                       wp_xsize;
153
                               pix_d
                                       wl_xsize;
154
                              pix_d
                                       wp_ysize;
155
                              pix_d
                                       wl_ysize;
156
                              pix_d
                                       wp_vxorig;
157
                              pix_d
                                       wl_vxorig;
158
                              pix_d
                                       wp_vyorig;
159
                                       wl_vyorig;
                              pix_d
160
                              pix_d
                                       wp_vxsize;
161
                              pix_d
                                       wl_vxsize;
162
                              pix_d
                                       wp_vysize;
163
                              pix d
                                       wl vysize;
                                       w_color;
164
                              short
165
                              sint
                                       w_border;
166
                                       wp_font;
                              char
167
                              char
                                       wl_font;
168
                              char
                                       w_curfont;
169
                              sint
                                       w_level;
170
                                       w_uboxes;
                              sint
171
                              cur_d
                                       w_xcur;
172
                              cur_d
                                       w_ycur;
173
                              pix_d
                                       w_xgcur;
174
                              pix_d
                                       w_ygcur;
175
                              sint
                                       w_tsig;
176
                              sint
                                       w_ntsig;
177
                              sint
                                       w_rsig;
178
                              sint
                                       w_csig;
179
                              word
                                       w_boxes;
180
                              uflags
                                       w_flags;
181
                              sint
                                       w_rstat;
182
                              union
183
                              {
184
                                       long
                                               W_XXX;
185
                              } w_pad;
186
                      };
```

187	The manifes of	the should be set on
188 1 <b>89</b>	ine meaning of	the structure members are:
190	wp_xorig	The x coordinate of the lower left corner of the
191		virtual screen relative to the lower left corner of
192		the screen. The coordinates are expressed in terms of
193		pixels. If the lower left corner is to the left of the
194		lower left corner of the screen, this value is
195		negative. This coordinate is used in portrait mode.
196	1	to the mental but had a laderer and
197 198	wl_xorig	As 'wp_xorig', but used in landscape mode.
199	wp_yorig	The y coordinate of the lower left corner of the
200		virtual screen in portrait mode.
201		
202	wl_yorig	As 'wp_yorig', but used in landscape mode.
203		The bands and adverse first of the state of the second of the
204	wp_xsize	The horizontal size of the virtual screen expressed in
205 206		pixels in portrait mode.
207	wl_xsize	As 'wp_xsize', but used in landscape mode.
208	W1_N3110	TO WE NOTE & Date and I'm Tanagoupe mode.
209	wp_ysize	The vertical size of the virtual screen expressed in
210	_	pixels in portrait mode.
211		
212	wl_ysize	As 'wp_ysize', but used in landscape mode.
213		The warrandings of the laws laft course of the
214 215	wp_vxorig	The x coordinate of the lower left corner of the window (excluding the border) relative to the lower
215		left corner of the virtual screen in portrait mode.
217		Tere corner or one virtual Sercen in portrate mode.
218	wl_vxorig	As 'wp_vxorig', but used in landscape mode.
219	<u>-</u>	· ·
220	wp_vyorig	The y coordinate of the lower left corner of the
221		window in portrait mode.
222	ul waria	As 'wp vyorig', but used in landscape mode.
223 224	wl_vyorig	AS WP_VYOTIS, but used in landscape mode.
225	wp_vxsize	The horizontal size of the window in portrait mode.
226	WF_1X0220	THE NOTES OF THE NAME OF THE POST OF THE P
227	wl_vxsize	As 'wp_vxsize', but used in landscape mode.
228		
229	wp_vysize	The vertical size of the window in portrait mode.
230	1	to lum numainal, but need in landanana made
231 232	wl_vysize	As 'wp_vysize', but used in landscape mode.
233	w_color	Background colour in the window (BLACK or WHITE).
234	<b>"_</b> 0020.	beenground colour in the wilder (colour of milese).
235	w_border	The type of the border:
236		NOBORDER - No border.
237		SLBORDER - Single line border.
238 239		DLBORDER - Double lines border. DSSSBORD - The left side is a double lines border and
239 2 <b>40</b>		the rest of the sides are single line
241		borders.
242		SDSSBORD - The right side is a double lines border and
243		the rest of the sides are single line
244		borders.
245		SSDSBORD - The upper side is a double lines border and
246 247		the rest of the sides are single line borders.
247 248		SSSDBORD - The lower side is a double lines border and
		TITLE THE ACTION OF A COMPTO TRICO POLICE MIN

249		the rest of the sides are single line
250		borders.
251		DDSSBORD - The left and right sides are double lines
252		borders and the upper and lower sides are
253		single line borders.
254		DSDSBORD - The left and upper sides are double lines
255		borders and the right and lower sides are
256		single line borders.
257		DSSDBORD - The left and lower sides are double lines
258		borders and the right and upper sides are
259		single line borders.
260		SDDSBORD - The right and upper sides are double lines
261		borders and the left and lower sides are
262		single line borders.
263		SDSDBORD - The right and lower sides are double lines
264		borders and the left and upper sides are
265		single line borders.
266		SSDDBORD - The upper and lower sides are double lines
267		
268		borders and the left and right sides are
		single line borders.
269		DDDSBORD - The lower side is a single line border and
270		the rest are double lines borders.
271		DDSDBORD - The upper side is a single line border and
272		the rest are double lines borders.
273		DSDDBORD - The right side is a single line border and
274		the rest are double lines borders.
275		SDDDBORD - The left side is a single line border and
276		the rest are double lines borders.
277		
278	wp_font	The initial font in portrait mode. The font can be in
279		.1 1.1 1.1
<b>…</b> / /		the range 'A' - 'Z'.
280		the range 'A' - 'Z'.
	wl_font	As 'wp_font', but used in landscape mode.
280 281	wl_font	•
280 281 282	_	As 'wp_font', but used in landscape mode.
280 281 282 283	wl_font w_curfont	•
280 281 282 283 284	w_curfont	As 'wp_font', but used in landscape mode.  The currently used font.
280 281 282 283 284 285	_	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will
280 281 282 283 284 285 286	w_curfont	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child
280 281 282 283 284 285 286 287	w_curfont	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special
280 281 282 283 284 285 286 287 288	w_curfont	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level
280 281 282 283 284 285 286 287 288 289	w_curfont	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special
280 281 282 283 284 285 286 287 288 289 290	w_curfont w_level	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.
280 281 282 283 284 285 286 287 288 289 290	w_curfont	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see
280 281 282 283 284 285 286 287 288 289 290 291	w_curfont w_level	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is
280 281 282 283 284 285 286 287 288 289 290 291 292 293	w_curfont w_level	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is significant only if the BX_USER flag in 'w_boxes' is
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294	w_curfont w_level	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is significant only if the BX_USER flag in 'w_boxes' is set (to be compatible with older versions of the
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295	w_curfont w_level	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is significant only if the BX_USER flag in 'w_boxes' is set (to be compatible with older versions of the window handler, it was done in this way). If BX_USER
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296	w_curfont w_level	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is significant only if the BX_USER flag in 'w_boxes' is set (to be compatible with older versions of the
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297	w_curfont w_level w_uboxes	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is significant only if the BX_USER flag in 'w_boxes' is set (to be compatible with older versions of the window handler, it was done in this way). If BX_USER is not set, this value is assumed to be zero.
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298	w_curfont w_level	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is significant only if the BX_USER flag in 'w_boxes' is set (to be compatible with older versions of the window handler, it was done in this way). If BX_USER is not set, this value is assumed to be zero.  x coordinate for the text cursor position. This is
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299	w_curfont w_level w_uboxes	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is significant only if the BX_USER flag in 'w_boxes' is set (to be compatible with older versions of the window handler, it was done in this way). If BX_USER is not set, this value is assumed to be zero.  x coordinate for the text cursor position. This is only used to return the initial position of the
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300	w_curfont w_level w_uboxes	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is significant only if the BX_USER flag in 'w_boxes' is set (to be compatible with older versions of the window handler, it was done in this way). If BX_USER is not set, this value is assumed to be zero.  x coordinate for the text cursor position. This is
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301	w_curfont w_level w_uboxes w_xcur	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is significant only if the BX_USER flag in 'w_boxes' is set (to be compatible with older versions of the window handler, it was done in this way). If BX_USER is not set, this value is assumed to be zero.  x coordinate for the text cursor position. This is only used to return the initial position of the cursor, which is the upper left corner of the window.
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302	w_curfont w_level w_uboxes	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is significant only if the BX_USER flag in 'w_boxes' is set (to be compatible with older versions of the window handler, it was done in this way). If BX_USER is not set, this value is assumed to be zero.  x coordinate for the text cursor position. This is only used to return the initial position of the
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303	w_curfont w_level w_uboxes w_xcur	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is significant only if the BX_USER flag in 'w_boxes' is set (to be compatible with older versions of the window handler, it was done in this way). If BX_USER is not set, this value is assumed to be zero.  x coordinate for the text cursor position. This is only used to return the initial position of the cursor, which is the upper left corner of the window.  y coordinate for the text cursor position.
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304	w_curfont w_level w_uboxes w_xcur	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is significant only if the BX_USER flag in 'w_boxes' is set (to be compatible with older versions of the window handler, it was done in this way). If BX_USER is not set, this value is assumed to be zero.  x coordinate for the text cursor position. This is only used to return the initial position of the cursor, which is the upper left corner of the window. y coordinate for the text cursor position. x coordinate for the graphic cursor. This one is
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305	w_curfont w_level w_uboxes w_xcur	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is significant only if the BX_USER flag in 'w_boxes' is set (to be compatible with older versions of the window handler, it was done in this way). If BX_USER is not set, this value is assumed to be zero.  x coordinate for the text cursor position. This is only used to return the initial position of the cursor, which is the upper left corner of the window.  y coordinate for the graphic cursor. This one is only used to return the initial position (which is
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306	w_curfont w_level w_uboxes w_xcur	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is significant only if the BX_USER flag in 'w_boxes' is set (to be compatible with older versions of the window handler, it was done in this way). If BX_USER is not set, this value is assumed to be zero.  x coordinate for the text cursor position. This is only used to return the initial position of the cursor, which is the upper left corner of the window. y coordinate for the text cursor position. x coordinate for the graphic cursor. This one is
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307	w_curfont w_level w_uboxes w_xcur w_ycur w_xgcur	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is significant only if the BX_USER flag in 'w_boxes' is set (to be compatible with older versions of the window handler, it was done in this way). If BX_USER is not set, this value is assumed to be zero.  x coordinate for the text cursor position. This is only used to return the initial position of the cursor, which is the upper left corner of the window.  y coordinate for the graphic cursor. This one is only used to return the initial position (which is the lower left corner of the window).
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308	w_curfont w_level w_uboxes w_xcur	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is significant only if the BX_USER flag in 'w_boxes' is set (to be compatible with older versions of the window handler, it was done in this way). If BX_USER is not set, this value is assumed to be zero.  x coordinate for the text cursor position. This is only used to return the initial position of the cursor, which is the upper left corner of the window.  y coordinate for the graphic cursor. This one is only used to return the initial position (which is
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307	w_curfont w_level w_uboxes w_xcur w_ycur w_xgcur	As 'wp_font', but used in landscape mode.  The currently used font.  The level of the window. A newly created window will be on level 0 if it is not a special and not a child window, and on the lowest level if it is a special window (see the SPECIAL flag), and on the top level of its window group if it is a child window.  The maximal number of user defined boxes allowed (see the Winubox() request). The value of this member is significant only if the BX_USER flag in 'w_boxes' is set (to be compatible with older versions of the window handler, it was done in this way). If BX_USER is not set, this value is assumed to be zero.  x coordinate for the text cursor position. This is only used to return the initial position of the cursor, which is the upper left corner of the window.  y coordinate for the graphic cursor. This one is only used to return the initial position (which is the lower left corner of the window).

311		when it has moved to the top level (level zero). If
312		O, no signal will be sent.
313		
314	w_ntsig	As above, but signals are sent when the window moves
315		from the top level to a lower level.
316		
317	w_rsig	The signal to be sent to the processes in the window
318		when the window has changed in some way. If O, no
319		signal will be sent.
320		
321	w_csig	The signal to be sent to the processes in the window
322	_	when the close box in the border is used. If O, no
323		signal is sent, instead all requests to this window
324		will be terminated with bad status.
325	_	
326	w_boxes	Contains flags indicating which boxes shall be present
327		in the border:
328		
329		BX_HSCR - Scroll left and right boxes and the
330		horizontal visible indicator shall be
331		present in the border.
332		BX_VSCR - Scroll up and down boxes and the vertical
333		visible indicator shall be present in the
334		border.
335		BX_CLOS - The close box shall be present in the
336		border.
337		BX_SIZE - The size box shall be present in the border.
338		BX_MOVE - The move box shall be present in the border.
339		BX_ZOOM - The zoom box shall be present in the border.
340		BX_AVIS - The scroll boxes and the horizontal and
341		vertical visible indicators are only visible
342		if the whole virtual screen is not visible.
343		BX_BLOW - The "blow up" box shall be present in the
344		border (see the Windflsz() request).
345		BX_HELP - The help box shall be present in the border
346		(see the Winhelp() request).
347		BX_USER - Indicates that the value of the 'w_uboxes'
348 349		member is significant.
350	u flage	Contains come floor.
351	w_flags	Contains some flags:
352		PMODE - Indicates that coordinates have been
353		given for portrait mode.
354		LMODE - Indicates that coordinates have been
355		given for landscape mode.
356		SAVETEXT - Save the text contents of the virtual
357		screen.
358		SAVEBITMAP - Save the bitmap contents of the virtual
359		screen (virtual bitmap) (reserved for
360		future use).
361		OVERLAP - The window is overlapped flag.
362		LOCK - The window is locked on the highest level
363		(level 0).
364		NOOVER - The window must not be overlapped by
365		another window.
366		NOCURSOR - Cursor not visible.
367		NOMOVE - The window must not be moved or change
368		size.
369		ALLSCR - The window must be the whole virtual
370		screen.
371		SPECIAL - A special window will be added on the
372		lowest level. Special windows are always

373		on lower levels than non-special windows
374		and their level does not change when the
375		level of other windows are changed. They
376		can for example be used as menu windows.
377	KEYSCROLL	- Every time a key is pressed it is checked
378	KETSCRULL	if the whole cursor is visible and if not
379		the window is scrolled.
380	WRITSCROLL	- After each write request to the window,
381		it is checked if the whole cursor is
382		visible and if not the window is
383		scrolled.
384	ALTMPNT	- Allocate space to store a mouse pointer
385		which is used when we point to this
386		window. Initially the mouse pointer will
387		be the same as the global pointer.
388		See the Winchmpnt() request.
	DEL ATTUE	· · · · · · · · · · · · · · · · · · ·
389	RELATIVE	- The coordinates 'w_xorig' and 'w_yorig'
390		are supposed to be relative to the lower
391		left corner of the parent in this window
392		group (see section 8).
393	NOCPIN	<ul> <li>Makes it impossible to copy text into</li> </ul>
394		this window using the text copy facility
395		of the window handler.
396	NOCPOUT	- Makes it impossible to copy text from
397	11001 001	this window using the text copy facility
398		of the window handler. Instead the status
399		of the middle mouse button is reported on
400		mouse position reports. Note that the
401		middle button is only reported if this
402		flag is set.
403	TXTSIZE	- The 'wp_xsize', 'wl_xsize', 'wp_ysize',
404		'wl_ysize', 'wp_vxorig', 'wl_vxorig',
405		'wp_vyorig', 'wl_vyorig', 'wp_vxsize',
406		'wl_vxsize', 'wp_vysize', and 'wl_vysize'
407		members are supposed to be given in term
408		of characters instead of pixels.
409		Note that in this case 'wp_vxorig',
410		'wl_vxorig', 'wp_vyorig', and 'wl_vyorig'
411		must be given relative to the upper left
412		corner of the virtual screen.
413	WGROUP	- This window shall belong to a window
414		group (see section 8).
415	REL_ULC	- This window shall follow its parent
416	-	window relative the upper left corner
417		of the parent (this flag has no effect
418		if the window is not a child window).
419	REL URC	- This window shall follow its parent
420	NLL_UNU	
		window relative the upper right corner
421		of the parent (this flag has no effect
422	DE: 110	if the window is not a child window).
423	REL_LLC	- This window shall follow its parent
424		window relative the lower left corner
425		of the parent (this flag has no effect
426		if the window is not a child window).
427	REL_LRC	- This window shall follow its parent
428	=	window relative the lower right corner
429		of the parent (this flag has no effect
430		if the window is not a child window).
431		I. ONC WINDOW IS NOT G CHILD WINDOW/.
432	Note that a	t most one of the flags DEL HILL DEL HID
		t most one of the flags REL_ULC, REL_URC,
433		REL_LRC may be set.
434	wit russe I	lags are single bits in the flags word.

```
435
                               Of these only the OVERLAP flag is non-significant when
436
                               creating a window.
437
                               All the remaining bits should be zero to quarantee
438
                               compatibility with future versions.
439
440
                w_rstat
                              Return status:
441
442
                              WE_ILPARA - an illegal parameter was specified.
443
                                       - the window can not be created because of
                              WE_LORO
444
                                           another window with the NOOVER or LOCK
445
                                           flag set.
446
                              WE ALRCR - the window has already been created.
447
                              WE_ALLSCR - the whole virtual screen is not visible
448
                                          and the ALLSCR flag is set.
449
                              WE_NOMEM - enough memory does not remain to create
450
                                           the window.
451
                              WE_FATHER - the window has the RELATIVE flag set, but
452
                                           there is no parent window.
453
                              WE_ILMOD - the coordinates for the current screen
                                           mode has not been given, e.g. the screen
454
455
                                           is in landscape mode and the LMODE flag
456
                                          is not set.
457
                              WE NOFONT - the specified default font can not be
458
                                          loaded.
459
              Of the above members, only the following are used when a window is
460
              created:
461
462
                      wp_xorig or wl_xorig, wp_yorig or wl_yorig, wp_xsize or
463
464
                      wl_xsize, wp_ysize or wl_ysize, wp_vxorig or wl_vxorig,
                      wp_vyorig or wl_vyorig, wp_vxsize or wl_vxsize, wp_vysize
465
                      or wl_vysize, w_color, w_border, wp_font or wl_font, w_tsig,
466
467
                      w_ntsig, w_rsig, w_csig, w_boxes, w_flags
468
              On exit the values of these members remains the same, except for some
469
470
              adjustments that may occur in order to make the window fit, etc.
              The other members have on exit received their initial values.
471
472
473
474
              6.2
                      Move Window to Level Zero
                      ------
475
476
              The level zero window is the window that receives the keyboard input.
477
478
              The request
479
480
                      Winlevel(fd, bp)
481
                      int
482
                      struct winlevel *bp;
483
              is used to move a window which does not belong to a window group to
484
              the zero level. If the window indicated by 'fd' belongs to a window
485
486
              group, the whole group is moved to the top without altering the
487
              relative levels inside the group.
              The winlevel structure looks like:
488
489
490
                      typedef char
                                      sint;
491
492
                      struct winlevel
493
494
                              sint
                                      l_rstat;
495
                              union
```

{

```
497
                                      long
                                              1_xxx;
498
                              } l_pad;
499
                      };
500
              where 'l_rstat' is the return status:
501
502
503
                                 - everything is well.
                                 - the window has not been created yet.
504
505
                      WE_SPECIAL - the window can not be moved to the top because
506
                                   it is a special window.
507
                      WE_LORO
                                 - the level can not be changed because of another
508
                                   window with the LOCK or NOOVER flags set.
509
510
511
              6.3
                      Move Window to the Top Level of its Window Group
                      512
513
514
              To move a window, belonging to a window group, to the top level of the
515
              group, use the request:
516
517
                      Winllev(fd, bp)
518
                      int
                                       fd:
519
                      struct winlevel *bp;
520
521
              'fd' is the file descriptor for the window and the winlevel structure
522
              was described in section 6.2.
523
524
525
                      Alter a Window
              6.4
                      -----
526
527
528
              To alter the size, position, etc. of a window, the request
529
530
                      Winalter(fd, bp):
531
                      int
                                      fd:
532
                      struct winstruc *bp;
533
534
              is used. If the window is a parent of a window group, all the
535
              children are also moved according to the flags REL_ULC, REL_URC,
536
              REL_LLC, and REL_LRC. If none of these flags are set for a child
537
              window, the child is not moved.
538
              The winstruc structure was described in section 6.1. On entry to this
539
              request, the following structure member values are significant:
540
541
                      wp_xorig or wl_xorig, wp_yorig or wl_yorig, wp_vxorig or
542
                      wl_vxorig, wp_vyorig or wl_vyorig, wp_vxsize or wl_vxsize,
543
                      wp_vysize or wl_vysize
544
545
              Further the PMODE and LMODE flags in 'w_flags' are used to check that
546
              the data is relevant and if the TXTSIZE flag is set, the coordinates
547
              and sizes are interpreted in units of characters. The size of the
548
              current default font is used.
              The remaining parameters can not be changed using this request, but
549
550
              the current values of them are returned.
              'w_rstat' is the return status:
551
552
553
                               - all is well.
                      W OK
554
                      WE_NOTCR - the window has not been created yet.
555
                      WE_ILPARA - an illegal parameter value was used.
556
                      WE_LORO
                               - the window can not be altered because of another
557
                                 window with the LOCK or NOOVER flags set.
```

WE\_ALLSCR - the whole virtual screen will not be visible and

559 the ALLSCR flag for the window is set. 560 WE NOMOVE - it is not allowed to change the location or the 561 size of the window (the NOMOVE flag is set). 562 WE ILMOD - data for the current screen mode is not present. 563 564 565 6.5 Alter a Window without Affecting Child Windows 566 -----567 This request is identical to the Winalter() request, except that if 568 569 the specified window is a parent of a window group, its child windows 570 are not moved. 571 The request is: 572 573 Win1alter(fd, bp) 574 int 575 struct winstruc \*bp; 576 577 578 6.6 Set up Default Size and Location for a Window 579 -----580 When the "blow up" box is used the size and location of the window 581 582 toggles between the default size and location and the size and location it had before it was altered to the default. 583 584 When a window is created, its initial default size and location will 585 be the same as the initial size and location of the window. When the default font is changed, the default size and location will 586 be set to the same as the size and location of the window after the 587 default font has been changed. 588 589 To set up another default size and location, use the request: 590 591 Windflsz(fd, bp) fd: 592 int 593 struct winstruc \*bp; 594 595 The winstruc structure was described in section 6.1. On entry to this 596 request the following structure members are significant: 597 598 wp\_xorig or wl\_xorig, wp\_yorig or wl\_yorig, wp\_vxorig or 599 wl\_vxorig, wp\_vyorig or wl\_vyorig, wp\_vxsize or wl\_vxsize, 600 wp\_vysize or wl\_vysize 601 Further the PMODE and LMODE flags in 'w\_flags' are used to check that 602 the data is relevant and if the TXTSIZE flag is set the coordinates 603 604 and sizes are interpreted in units of characters. The size of the 605 current default font is used. The return status - 'w\_rstat' - is: 606 607 608 - all is well. WE NOTCR - the window has not been created yet. 609 WE\_ILMOD - data for the current screen mode is missing. 610 WE\_ILPARA - an illegal value was specified. 611 612 613 614 6.7 Alter Window Flags 615 -----616 To alter the flags in the 'w\_flags' word for a window, use the 617 618 request: 619

Winflags(fd, bp);

```
621
                      int
                                       fd:
622
                      struct flgstruc *bp;
623
624
              The flastruc structure looks like:
625
626
                      typedef unsigned long
                                              uflags:
627
                      typedef
                                       char
                                              sint:
628
629
                      struct flgstruc
630
                      1
631
                                      f flags;
                              uflags
632
                              sint
                                      f_rstat;
633
                              union
634
635
                                      long
                                              f xxx;
636
                              } f pad;
637
                      };
638
              'f_flags' is the new flags for the window.
639
640
              The following flags may be altered: LOCK, NOOVER, NOCURSOR, NOMOVE,
641
              ALLSCR, KEYSCROLL, WRITSCROLL, NOCPIN, NOCPOUT, REL_ULC, REL_URC,
642
              REL LLC, and REL LRC.
643
              The following flags are ignored: PMODE, LMODE, SAVETEXT, SAVEBITMAP,
644
              OVERLAP, SPECIAL, ALTMPNT, RELATIVE, TXTSIZE, and WGROUP.
645
              The the bits not used in the flags word should be zero to guarantee
646
              compatibility with future versions.
647
              'f_rstat' is the return status:
648
649
                                - everything is OK.
                                - the flags can not be altered in this way because
650
651
                                  the window is overlapped or it is not on the top
                                  level.
652
653
                      WE_ALLSCR - the whole virtual screen is not visible and the
654
                                  ALLSCR flag was set.
655
656
657
              6.8
                      Get Window Status
658
                      -----
659
              To get the current status of a window, use the request
660
661
662
                      Winstat(fd, bp);
663
                      int
664
                      struct winstruc *bp;
665
666
              The winstruc structure was described in section 6.1.
              On exit all the members are set to their current values. Only one of
667
668
              portrait or landscape mode coordinates and font is returned,
              depending on the mode of the screen. Which one is indicated by the
669
              PMODE and LMODE flags.
670
671
              The return status 'w_rstat' is:
672
673
                               - all is well.
674
                      WE_NOTCR - the window has not been created yet.
675
676
             6.9
677
                      Insert a Header in a Window Border
678
                      ------
679
680
              To insert a header, such as the program name, in the border of a
681
              window, use the request
```

```
Winheader(fd, bp);
683
684
                                        fd;
685
                       struct headstruc *bp:
686
              where the headstruc structure looks like:
687
688
                      typedef unsigned short word:
689
690
691
                      struct headstruc
692
                       {
693
                               char
                                       h_hdr[HDRSIZE];
694
                               word
                                       h_flags;
695
                               union
696
697
                                       long
                                               h_xxx;
698
                               } h_pad;
699
                      };
700
701
              'h_hdr[]' is the header string, 'h_flags' contains some flags:
702
703
                      H_INVHD - Invert the window header (relative the window
704
                                  background).
705
                      H_INVTOP - Invert the top window header (relative H INVHD).
706
707
              The remaining bits should be zero to guarantee compatibility with
708
              future versions.
709
              Note that the header can be added before the window is created.
710
711
712
              6.10
                      Icon Support
713
                      -----
714
715
              The window handler can automatically take care of decoding commands
              given by first pointing to an icon, menu item, etc. and then pressing
716
717
              an appropriate key on the mouse or the keyboard.
718
              The request
719
720
                      Winicon(fd, bp);
721
                      int
722
                      struct winicon *bp;
723
              is used to specify that when the pointer points inside a specified
724
725
              area in the window, a specified code sequence shall be sent to the
726
              calling process by putting it in the keyboard input buffer for the
727
              window.
728
              The winicon structure looks like:
729
730
                      typedef
                                        short pix_d;
731
                      typedef unsigned short word;
732
                      typedef
                                        char
                                               sint;
733
734
                      struct winicon
735
                      {
736
                              pix_d
                                      ip_xorig;
737
                              pix_d
                                      il_xorig;
                                      ip_yorig;
738
                              pix_d
739
                              pix_d
                                      il_yorig;
740
                              pix_d
                                      ip_xsize;
741
                              pix_d
                                      il_xsize;
742
                              pix_d
                                      ip_ysize;
743
                              pix_d
                                      il_ysize;
744
                              char
                                      i_cmdseq[ICONSEQLEN];
```

806

```
745
                               word
                                        i_flags;
746
                                sint
                                        i_rstat;
747
                               union
748
                                1
749
                                        long
                                                i_xxx;
750
                                } i_pad;
751
                       };
752
753
               'ip_xorig', 'il_xorig' and 'ip_yorig', 'il_yorig' is the lower left
754
               corner of the area relative to the lower left corner of the virtual
              screen in portrait and landscape mode, respectively. 'ip_xsize', 'il_xsize' and 'ip_ysize', 'il_ysize' is the width and height of the
755
756
757
              area in portrait and landscape mode, respectively.
758
               'i cmdseq[]' is the sequence to be sent to the calling process (it
759
              can be of zero length).
               'i_flags' contains some flags indicating the type of icon and some
760
761
              attributes:
762
763
                                 - Portrait mode coordinates are given.
                       I PMODE
                                  - Landscape mode coordinates are given.
764
765
                       I PRESS
                                 - Send the sequence when the mouse pointer points to
766
                                    the area and the left button is pressed.
767
                       I RELEASE - Send the sequence when the mouse pointer points to
                                    the area and the left button is released.
768
                       I INVERT - Invert the area occupied by the icon when the
769
770
                                    mouse pointer is pointing to it.
                                 - The sequence is sent when the mouse pointer moves
771
                       I_ENTER
772
                                    into the area. The area does not have to be
                                   visible. The I_INVERT flag is ignored.
773
774
                                  - As I_ENTER but the sequence is sent when we leave
                       I_LEAVE
775
                                    the area.
                       I_REMOVE - The icon is removed when the sequence has been
776
777
778
                                   The sequence is sent only if there is a pending
                       I_RQST
779
                                   read request to the window.
                       I_SETCHK - When I_ENTER and/or I_LEAVE is set, it is checked
780
781
                                   if the mouse pointer is inside or outside,
782
                                    respectively, the specified area, and if so the
783
                                   sequence is sent immediately.
784
                                 - The sequence is sent only if it is the level zero
                       I_LZER0
785
786
                                  - The coordinates and sizes of the icon is supposed
                       I_TEXT
787
                                    to be given in term of characters instead of
788
                                    pixels. Note that 'ip_xorig' and 'ip_yorig' or
789
                                    'il_xorig' and 'il_yorig' in this case are inter-
790
                                    preted as the character position relative the
791
                                   upper left corner of the virtual screen.
792
                                    When the default font is changed, the locations
793
                                   and sizes of icons set up with this flag set are
794
                                   adjusted.
795
796
              The remaining bits should be zero to guarantee compatibility with
797
              future versions.
798
              Note that if no one of I PRESS, I RELEASE, I ENTER, or I LEAVE is
799
              given, I_PRESS is assumed. I_ENTER and I_LEAVE overrides I_PRESS and
800
              I_RELEASE.
801
802
              The return status 'i_rstat' is:
803
804
                       W OK
                                 - everything is well.
```

WE\_NOTCR - the window is not created yet.

WE\_ILPARA - any of the input parameters are illegal.

```
807
                       WE NOICON - no memory left for the new icon.
808
                       WE_ONICON - the icon will come above another icon in the same
809
                                   window.
810
                       WE_ILMOD - no coordinates are given for the current screen
811
                                   mode.
812
813
              6.11
                       Remove Icon's
814
                       -----
815
816
              To remove all set up icon's for a window, use the request:
817
818
                       Rmicons(fd):
819
                       int
                               fd:
820
821
822
              6.12
                      Mouse Substitute Keys
823
                       -----
824
825
              To make it possible to use the window handler without a mouse, the
826
              different functions supported by the mouse can be simulated by
827
              function and other special keys on the ABC99 keyboard (these keys
828
              generates codes with the most significant bit set).
829
              To specify these keys, use the request:
830
831
                      Winmsub(fd, bp);
                                       fd:
832
                      int
833
                      struct substit *bp;
834
              The file descriptor used must be the one obtained when the window
835
              handler was activated (the first open request to the handler). The
836
837
              structure substit looks like:
838
839
                      typedef char
840
841
                      struct substit
842
                      {
843
                              sint
                                               c_initflg;
                              unsigned char
844
                                               c_keys[SUBSTKEYS];
845
                              unsigned char
                                               c_step;
846
                              unsigned char
                                               c_lstep;
                              union
847
848
849
                                      long
                                               C_XXX;
850
                              } c_pad;
                      };
851
852
853
              The meaning of the different members are:
854
855
                c_initflg
                              If ON the mouse simulation keys will be enabled after
856
                              this request. If OFF they will initially be disabled.
857
                c_keys[]
                              The keys used as substitue for the mouse.
858
                c_step
                              Step for normal mouse pointer move (no. of pixels).
859
                c_lstep
                              Step for long mouse pointer move (no. of pixels).
860
              The index for the different keys in the 'c_keys[]' array are:
861
862
863
                S_ONOFF
                              The key used to toggle the mouse simulation keys on or
864
                              off. When off, the keys behaves as normal (except
865
                              'S_ONOFF').
866
                S_MPU
                              Move mouse pointer up.
                S_MPD
                              Move mouse pointer down.
867
868
                              Move mouse pointer left.
                S_MPL
```

```
S_MPR
                              Move mouse pointer right.
869
870
                S_MPUL
                             Move mouse pointer up - left.
                              Move mouse pointer up - right.
                S MPUR
871
872
                S MPDL
                             Move mouse pointer down - left.
873
                S MPDR
                              Move mouse pointer down - right.
874
                             Move mouse pointer up long.
                S LMPU
                             Move mouse pointer down long.
875
                S LMPD
               S_LMPL
876
                             Move mouse pointer left long.
877
                             Move mouse pointer right long.
                S_LMPR
878
                S_LMPUL
                             Move mouse pointer up - left long.
879
                S LMPUR
                             Move mouse pointer up - right long.
880
                S_LMPDL
                             Move mouse pointer down - left long.
881
                S LMPDR
                              Move mouse pointer down - right long.
               S_PCMD
                             Point to command key (replaces the left key on the
882
883
                             Change window level key (replaces the right key on the
884
               S CHWIN
885
                              mouse).
886
               S_MCA
                             Mark text area to copy (replaces the middle key on the
887
                              mouse).
888
889
              Pressing and releasing a button on the mouse is replaced by pressing
890
              the chosen keyboard key twice.
              Note that no keys will be occupied by these keys if this request has
891
892
              not been issued.
893
894
895
              6.13
                     Alter the Background Pattern
896
                      ------
897
898
              To alter the pattern of the background, use the request:
899
900
                     Winchbg(fd, bp)
901
                      int
                                        fd:
902
                     struct chbgstruc *bp;
903
              'fd' must be the file descriptor obtained when the window handler was
904
905
              activated.
              The chbgstruc structure looks like:
906
907
                     typedef unsigned short word;
908
909
910
                     struct chbastruc
911
                      ſ
912
                             word
                                     cb_bitmap[BGPSIZE];
913
                             union
914
915
                                     long
                                             cb_xxx;
916
                             } cb_pad;
917
                     };
918
919
              'cb_bitmap[]' is the bit pattern of a 16 x BGPSIZE pixels area which
              will be repeated all over the background.
920
921
              Note that the most significant bit in a "word" is displayed to the
              left on the screen.
922
923
924
              6.14
925
                     Get the Visible Parts of a Window or the Background
926
                     -----
927
928
              To get the visible parts of a window or the background, use the
              request:
929
```

```
931
                       Wingetvis(fd, bp, bc)
932
                                       fd:
933
                       struct buffer *bp:
934
                       int
935
936
              'fd' is the file descriptor for the window, or the file descriptor
              obtained when the window handler was activated if the visible parts
937
938
              of the background are desired.
939
              'bc' is the size of the buffer structure.
940
              The buffer structure looks like:
941
942
                      struct buffer
943
944
                              struct visdes v;
945
                               struct rectdes b[VSIZE];
946
                      };
947
948
              The visdes structure is a parameter structure and looks like:
949
950
                      typedef char
                                       sint;
951
                      struct visdes
952
953
                      {
954
                              short
                                       v nrect:
955
                              sint
                                       v_rstat;
956
                              union
957
958
                                      long
                                               V_XXX;
959
                               } v pad;
960
                      };
961
962
              The rectdes structure describes one rectangle which the visible part
963
              of the virtual screen or the background can be divided into:
964
965
                      typedef short pix_d;
966
967
                      struct rectdes
968
                      {
969
                              pix d
                                      r_xorig;
970
                              pix_d
                                      r_yorig;
971
                              pix_d
                                      r_xsize;
972
                              pix_d
                                     r_ysize;
973
                      };
974
975
              where 'r_xorig' and 'r_yorig' are the x and y coordinates respectively
976
              of the lower left corner of the rectangle. 'r_xsize' and 'r_ysize' are
977
              the width and height, respectively, of the rectangle.
978
              When this request is executed the 'v_nrect' member of visdes should
979
              contain the number of rectdes structures (VSIZE) in the buffer
              structure. The request returns the actual number of rectangles that
980
              the virtual screen (or the background) can be divided into in
981
982
              'v_nrect'.
              'v_rstat' is the return status:
983
984
985
                      W_OK
                             - 0k
986
                      WE_NOTCR - The window has not been created yet.
987
                      WE_SPACE - Not enough space to hold the rectangles (i.e. VSIZE
988
                                  is too small).
989
990
991
                      Inverse Video
```

6.15 Inverse Video

```
993
 994
               The request:
 995
 996
                       Winivideo(fd)
 997
                       int
                               fd:
 998
 999
               changes the screen to inverse video. 'fd' must be the file descriptor
               obtained when the window handler was activated.
1000
1001
1002
1003
               6.16
                      Normal Video
                       -----
1004
1005
1006
               The request:
1007
1008
                      Winnvideo(fd)
1009
                       int
                               fd;
1010
               restores the screen to normal video. 'fd' must be the file descriptor
1011
1012
               obtained when the window handler was activated.
1013
1014
1015
               6.17
                      Make the Cursor Visible in the Window
                       1016
1017
1018
               To make the cursor visible in the window, use the request:
1019
1020
                      Wincurvis(fd)
                       int
                              fd;
1021
1022
               If the whole cursor is not visible, the window is scrolled.
1023
1024
1025
1026
               6.18
                      Change Mouse Pointer
                       ______
1027
1028
1029
               To change the layout of the mouse pointer, use the request:
1030
1031
                      Winchmpnt(fd, bp)
                       int
1032
                                       fd:
1033
                      struct npstruc *bp;
1034
1035
               If 'fd' is the file descriptor obtained when the window handler was
               activated, the global mouse pointer is altered. Otherwise the mouse
1036
1037
               pointer for the window indicated by the file descriptor is altered
1038
               (in this case, the ALTMPNT flag for the window must be set).
1039
               The npstruc structure looks like:
1040
1041
                       typedef
                                       short pix d;
1042
                       typedef unsigned long
                                              dword:
                      typedef unsigned char
1043
                                              byte:
1044
                       typedef
                                              sint;
                                       char
1045
1046
                      struct npstruc
1047
                       {
1048
                              pix_d
                                      np_xsize;
1049
                              pix_d
                                      np_ysize;
1050
                              pix_d
                                      np_xpnt;
1051
                                      np_ypnt;
                              pix_d
                                      np_and[MPSIZE];
1052
                              dword
1053
                               dword
                                      np_or[MPSIZE];
```

np\_flags;

byte

-----

```
1055
                                sint
                                        np_rstat;
1056
                                union
1057
1058
                                        long
                                                np_xxx;
1059
                                } np_pad;
1060
                        };
1061
1062
               'np_xsize' and 'np_ysize' are the width and height, respectively, of
1063
               the new mouse pointer. The maximal width is 32 pixels and the height
1064
               MPSIZE pixels.
               'np_xpnt' and 'np_ypnt' are the pixel which is the pointing part of
1065
               the mouse pointer. It shall be specified relative the upper left
1066
1067
               corner of the mouse pointer.
               'np_and[]' and 'np_or[]' are masks used to construct the mouse
1068
1069
1070
               Each pixel row of the mouse pointer is constructed by the operation:
1071
1072
                       (x & np_and[prow]) ! np_or[prow]
1073
1074
               where 'x' is the contents of the graphic memory. Note that the most
               significant bit in a "dword" is displayed to the left on the screen.
1075
               'np flags' is reserved for future use and should be zero to guarantee
1076
1077
               compatibility with future versions.
1078
               'np_rstat' is the return status:
1079
                                 - Ok.
1080
                       M OK
1081
                       WE_ILPARA - An illegal value was specified.
                       WE_NOTCR - The window has not been created yet.
1082
1083
                                - The ALTMPNT flag for the window is not set, and
                                   therefore the mouse pointer can not be changed.
1084
1085
1086
1087
               6.19
                       Get Number of Open Windows
                       ______
1088
1089
1090
               To find out how many windows which are open and/or created, use the
1091
               request:
1092
1093
                       Wincht(fd, bp)
                       int
1094
                                       fd:
1095
                       struct nwstruc *bp;
1096
1097
               'fd' is the file descriptor obtained when the window handler was
               activated or the file descriptor for a window.
1098
1099
               The nwstruc structure looks like:
1100
                       struct nwstruc
1101
1102
1103
                               short
                                       nw open;
1104
                               short
                                       nw created;
1105
                               union
1106
1107
                                       long
                                               nw_xxx;
1108
                               } nw_pad;
1109
                       };
1110
               'nw_open' is the number of windows currently open and 'nw_created' is
1111
1112
               the number of windows currently created (and opened).
1113
1114
1115
               6.20
                       Restore Screen
```

```
1117
1118
               To restore the screen, i.e. rewrite the whole screen, use the request:
1119
1120
                       Winrestor(fd)
1121
                       int
                               fd:
1122
1123
               'fd' must be the file descriptor obtained when the window handler was
1124
               activated.
1125
1126
1127
               6.21
                      Get Text Contents of Window
1128
                       1129
1130
               To get the text contents of a window, use the request:
1131
1132
                      Wingettxt(fd, bp, bc)
1133
                       int
                                      fd;
1134
                       struct buffer
                                      *bp;
1135
                       int
                                      bc;
1136
1137
               'fd' is the file descriptor for the window. The structure buffer
               consists of a parameter structure followed by a buffer with space
1138
1139
               to hold the desired text contents:
1140
                      struct buffer
1141
1142
1143
                              struct txtstruc s;
1144
                                               b[BSIZE];
                              char
                      };
1145
1146
1147
              The txtstruc structure looks like:
1148
1149
                      typedef short
                                      cur_d;
                       typedef char
1150
                                      sint;
1151
1152
                      struct txtstruc
1153
1154
                              cur_d
                                      tx_row;
                                      tx_col;
1155
                              cur_d
1156
                              cur_d
                                      tx rcnt;
1157
                              cur_d
                                      tx_ccnt;
1158
                              sint
                                      tx_rstat;
1159
                              union
1160
1161
                                      long
                                              tx_xxx;
1162
                              } tx_pad;
1163
                      };
1164
               'tx_row' is the row number of the first row to be read and 'tx_col'
1165
               the number of the first column. 'tx_rcnt' and 'tx_ccnt' is the number
1166
              of rows and columns, respectively, to be read. BSIZE must be at least
1167
1168
               tx_rcnt * tx_ccnt.
1169
               'tx_rstat' is the return status:
1170
1171
                                - Everything is ok.
1172
                      WE_TSAVE - The text contents of the window is not saved.
1173
                      WE_ILPARA - Illegal parameters was given.
1174
1175
1176
              6.22
                      Test if Window Handler is Activated
                      ------
1177
1178
```

```
1179
               To test if the window handler is activated, use the request:
1180
                       Wintest(fd)
1181
1182
                       int
1183
1184
               'fd' is the file descriptor for a window or the one obtained when
1185
               the handler was activated.
1186
               If a negative value is returned, the window handler is not present.
1187
1188
               6.23
1189
                       Set Initial Driver and Terminal Parameters
                       1190
1191
               This request is used to set the initial driver and terminal parameters
1192
               for windows. The request is:
1193
1194
1195
                       Winsinit(fd, bp)
1196
                       int
                                       fd:
1197
                       struct wininit *bp;
1198
1199
               'fd' must be the file descriptor obtained when the window handler was
1200
               activated.
1201
               The wininit structure looks like:
1202
1203
                       typedef unsigned long
                                              t_stop;
                       typedef unsigned short word;
1204
1205
1206
                       struct wininit
1207
                               t stop td tbstop[TSTOPSIZE];
1208
1209
                               word
                                       td term;
                               struct
1210
1211
                                       unsigned short c_iflag;
1212
1213
                                       unsigned short c_oflag;
1214
                                       unsigned short c_cflag;
                                       unsigned short c_lflag;
1215
                                       char
1216
                                                       c line;
1217
                                       unsigned char
                                                      c_ccs[8];
1218
                               } td_driver;
1219
                               union
1220
1221
                                       long
                                               td_xxx;
1222
                               } td_pad;
1223
                       };
1224
               'td_tbstop[]' contains the tab stops. A set bit indicates a tab stop.
1225
               The least significant bit of the first element corresponds to the
1226
               first character position of a row.
1227
               'td_term' contains initial VT-100 terminal flags:
1228
1229
1230
                       TD_NL
                                       linefeed newline mode.
1231
                       TD_WRAP
                                       auto wrap mode.
1232
                       TD_ORIGIN
                                       origin mode.
1233
                       TD_USCORE
                                       underscore character attribute.
                                       reverse character attribute.
                       TD_REVERSE
1234
1235
                       TD_SCREEN
                                       screen mode.
1236
                       TD_CUNDER
                                       underline cursor.
1237
                       TD_NONBLNK
                                       non-blinking cursor.
1238
                       TD_PHASE
                                       phased pattern mode.
1239
                       TD_NOSCR
                                       no scroll (page) mode.
```

The remaining bits in 'td\_term' should be zero to guarantee compatibility with future versions.

'td\_driver' is a structure which contains the driver parameters. It is the same structure as the termio structure (see the header file (sys/termio.h) and the documentation for the ioctl() unix system call).

The default tab stops are every eight position, of the terminal flags the TD\_WRAP flag is set by default, and the driver parameters are the same as those of the console when the window handler was activated.

6.24 Get Initial Driver and Terminal Parameters

To get the values of the initial driver and terminal parameters, use the request:

```
Winginit(fd, bp)
int fd;
struct wininit *bp;
```

'fd' must be the file descriptor obtained when the window handler was activated.

6.25 Set Up a Zoom List for a Window

A zoom list is a list of fonts to change between when the mouse pointer points to the zoom box and the left button of the mouse is pressed. Every time this happens, the next font in the zoom list becomes the default font for the window. When the end of the list is reached, the next font will be the first one in the list. When a zoom list is set up, the current default font will become the first font in the list followed by the fonts specified in the zoomlst structure.

To set up a zoom list, use the request:

'fd' is the file descriptor for the window. The zoomlst structure looks like:

```
typedef unsigned char
                         byte;
typedef
                  char
                         sint;
struct zoomlst
{
                 zp_list[Z00MSIZE];
        char
        char
                 zl_list[Z00MSIZE];
        byte
                 z_flags;
        sint
                 z_rstat;
        union
        {
                 long
                         Z_XXX;
        } z_pad;
};
```

'zp\_list[]' is the list of fonts to be used in portrait mode and 'zl\_list[]' is used in landscape mode.

```
1303
                'z_flags' contains some flags:
1304
1305
                       Z_PMODE - Portrait mode zoom list is given.
1306
                       Z_LMODE - Landscape mode zoom list is given.
1307
               The remaining bits should be zero to guarantee compatibility with
1308
1309
               future versions.
1310
               The return status 'z_rstat' is:
1311
1312
                                 - everything is ok.
1313
                       WE_ILPARA - an illegal font was specified.
1314
                       WE_ILMOD - no list is given for the current screen mode.
1315
1316
               Note that this request can be used before the window has been created.
1317
1318
1319
               6.26
                       Change the Default Font for a Window
1320
                       1321
1322
               To change the default font for a window, use the request:
1323
1324
                       Winndchr(fd, bp);
1325
1326
                       struct dfltchr *bp;
1327
1328
               'fd' is the file descriptor for the window and the dfltchr structure
1329
               looks like:
1330
                       typedef
1331
                                        short cur d;
1332
                       typedef unsigned char
                                               byte:
1333
1334
                       struct dfltchr
1335
1336
                                       dcp_font;
                               char
                                       dcl_font;
1337
                               char
                                       dcp_x;
1338
                               cur_d
1339
                               cur_d
                                       dcl_x;
                               cur_d
1340
                                       dcp_y;
1341
                               cur_d
                                       dcl_y;
1342
                               byte
                                       dc_rstat;
1343
                               union
1344
1345
                                       long
                                               dc_xxx;
                               } dc_pad;
1346
1347
                       };
1348
               'dcp_font' and 'dcl_font' are the new default font in portrait and
1349
               landscape mode, respectively. If the specified font is zero, the next
1350
1351
               font in the zoom list is used.
               'dcp_x', 'dcp_y', 'dcl_x', and 'dcl_y' is the character coordinates in
1352
1353
               portrait and landscape mode, respectively, for the middle character in
1354
               the window after the default font has been changed.
               'dc_flags' contains some flags:
1355
1356
1357
                       Z_PMODE - Data has been given for portrait mode.
1358
                       Z_LMODE - Data has been given for landscape mode.
1359
1360
               The remaining bits should be zero to guarantee compatibility with
1361
               future versions.
1362
               'dc_rstat' is the return status:
1363
1364
                       M OK
                                 - everything is ok.
```

```
1365
                       WE_NOTCR - the window has not been created yet.
1366
                        WE ILMOD - no data is given for the current screen mode.
                       WE ILPARA - an illegal font and/or illegal character
1367
1368
                                    coordinates were given.
                       WE TSAVE - the text contents of the virtual screen is not
1369
1370
                                    saved.
                       WE_ALLSCR - the ALLSCR flag for the window is set.
1371
                       WE_NOMOVE - the NOMOVE flag for the window is set.
1372
                       WE_NOFONT - the specified font does not exist.
1373
1374
1375
               This request does not (if possible) change the size of the window.
               The size of the virtual screen is however adjusted so it contains the
1376
               same number of character rows and columns.
1377
1378
1379
               6.27
1380
                       Turn the Screen
1381
                       -----
1382
1383
               To turn the screen from portrait to landscape mode or vice versa, use
1384
               the request:
1385
1386
                       Winturn(fd, bp)
1387
                       int
                                         fd:
                       struct modstruc *bp;
1388
1389
               All channels, except the one obtained when the window handler was
1390
               activated, must be closed.
1391
1392
               'fd' must be the file descriptor obtained when the window handler was
               activated. The modstruc structure looks like:
1393
1394
                       typedef char
1395
                                       sint:
1396
                       struct modstruc
1397
                       {
1398
                               sint
                                       m mode;
1399
1400
                               sint
                                        m_rstat;
1401
                               union
1402
                                {
1403
                                        long
                                                m XXX;
1404
                               } m pad;
1405
                       };
1406
               'm_mode' will on return be M_PORT if the new mode is portrait mode or
1407
1408
               M LAND if it is landscape.
1409
               'm_rstat' is the return status:
1410
1411

    everything is ok.

                       WE_OPEN - there are windows open.
1412
1413
1414
1415
               6.28
                       Get Screen Mode
1416
1417
               To get the current screen mode (portrait or landscape), use the
1418
1419
               request:
1420
1421
                       Winmode(fd, bp)
1422
                                         fd;
                       int
1423
                       struct modstruc *bp;
1424
               'fd' is the file descriptor obtained when the window handler was
1425
1426
               activated or the file descriptor for a window. The modstruc structure
```

1488

request:

```
1427
                was described in section 6.27. The 'm_mode' member contains the
1428
               current mode (M_PORT or M_LAND) and 'm_rstat' is always W_OK.
1429
1430
1431
               6.29
                       Add a User Defined Box
1432
                       -----
1433
1434
               User defined boxes are 16x16 pixels boxes in the left side of the
1435
               window border. When the mouse pointer points to a user box and the
1436
               left mouse button is pressed, a signal is sent to the process(es)
1437
               running in the window.
1438
               When a window is created, the maximal number of user defined boxes
1439
               for the window must be specified (see the Wincreat() request).
1440
               To set up a user defined box, use the request:
1441
1442
                       Winubox(fd, bp)
                                       fd;
1443
                       int
                       struct userbox *bp;
1444
1445
               'fd' is the file descriptor for the window. The userbox structure
1446
1447
               looks like:
1448
1449
                       typedef unsigned short
                                              word:
1450
                       typedef unsigned char
                                               byte;
1451
                       typedef
                                        char
                                               sint;
1452
1453
                       struct userbox
1454
1455
                               word
                                       bx_bmap[UBOXSIZE];
1456
                               short
                                      bx sig;
1457
                               byte
                                       bx_flags;
1458
                               sint
                                       bx rstat:
1459
                               union
1460
                               {
1461
                                       long
                                               bx_xxx;
1462
                               } bx_pad;
                       };
1463
1464
               'bx bmap[]' contains the bitmap for the box. Note that the most
1465
               significant bit in a "word" is displayed to the left on the screen.
1466
1467
               'bx sig' is the signal to be sent when the box is used.
               'bx_flags' is reserved for future use and should be zero to guarantee
1468
               compatibility with future versions.
1469
               'bx_rstat' is the return status:
1470
1471
1472
                                - all is well.
                       WE_NOTCR - the window has not been created yet.
1473
1474
                       WE_SPACE - the maximal number of user defined boxes have
1475
                                   already been set up.
1476
                       WE_ILPARA - an illegal signal number was specified.
1477
1478
1479
               6.30
                       Alter Help Box Sequence
1480
                       -----
1481
               The help box is a box in the upper side of the border containing a
1482
1483
               question mark which when used puts a character sequence on the key-
1484
               board input buffer. The intention is that all programs use this
1485
               facility so help can be requested in a similar manner in all programs.
1486
               When a window is opened, the help box sequence is initialized to a
```

single question mark (?). To alter this to another sequence, use the

```
1489
1490
                        Winhelp(fd, bp)
1491
                        int
                                        fd:
                        struct helpst *bp;
1492
1493
               'fd' is the file descriptor for the window. The helpst structure looks
1494
1495
1496
1497
                        typedef unsigned short word:
1498
1499
                       struct helpst
1500
1501
                                char
                                        hlp_seq[HLPSIZE];
1502
                                word
                                        hlp_flags;
1503
                               union
1504
1505
                                        long
                                               hlp_xxx;
1506
                                } hlp_pad;
1507
                       };
1508
1509
               'hlp_seq[]' is the new help box sequence. 'hlp_flags' is reserved for
               future use and should be zero to guarantee compatibility with future
1510
               versions of the window handler.
1511
1512
               Note that the help box sequence can be altered before the window has
1513
               been created.
1514
1515
1516
               6.31
                       Keyboard Input Signal
1517
                       -----
1518
1519
               To make it possible to know when there is something to read from the
1520
               keyboard buffer, a signal can be set up for this purpose. The signal
               will be sent when there is no pending read request to the window and
1521
1522
               reading the keyboard buffer will not lead to wait.
1523
               The request is:
1524
1525
                       Winkysig(fd, bp)
1526
                       int
                                        fd;
1527
                       struct kysigst *bp;
1528
1529
               'fd' is the file descriptor for the window and the kysigst structure
1530
               looks like:
1531
1532
                       struct kysigst
1533
                       {
1534
                                       ks_sig;
                               sint
1535
                               byte
                                       ks_flags;
1536
                               sint
                                       ks_rstat;
1537
                               union
1538
1539
                                       long
                                               ks_xxx;
1540
                               } ks_pad;
1541
                       };
1542
1543
               'ks_sig' is the signal to be sent. If zero, no signals are sent.
1544
               'ks_flags' is reserved for future use and should be zero to guarantee
1545
               compatibility with future version.
1546
               'ks_rstat' is the return status:
1547
1548
                                 - everything is well.
1549
                       WE_ILPARA - an illegal signal was specifiead.
1550
```

```
1551
1552
               6.32
                       Read the Contents of the Picture Memory
1553
                       1554
1555
               To read the contents of the picture memory for a window or the whole
1556
               screen, use the request:
1557
1558
                       Wpictrd(fd, bp, bc);
1559
                       int
                                       fd;
1560
                       struct buffer *bp;
1561
                       int
                                       bc;
1562
               'fd' is the file descriptor for the window or, if the contents of the
1563
               whole screen is desired, the file descriptor obtained when the window
1564
1565
               handler was activated. The buffer structure consists of a parameter
1566
               structure followed by a buffer big enough to hold the contents of the
1567
               specified picture memory area:
1568
1569
                       typedef unsigned char
                                              byte;
1570
1571
                      struct buffer
1572
1573
                               struct wpictblk p;
                                               b[BSIZE];
1574
                               byte
                      };
1575
1576
               The wpictblk structure looks like:
1577
1578
1579
                       typedef short pix_d;
1580
1581
                      struct wpictblk
1582
                       {
1583
                              pix_d
                                      p_xaddr;
1584
                               pix_d
                                      p_yaddr;
1585
                              pix_d
                                      p_width;
1586
                              pix_d
                                      p_height;
1587
                              union
1588
                               {
1589
                                      long
                                              P_XXX;
1590
                               } p pad;
1591
                      };
1592
               'p_xaddr' and 'p_yaddr' are the x and y pixel coordinates,
1593
              respectively, of the lower left corner of the area to read. 'p_width'
1594
              is the pixel width of the area and 'p_height' the pixel height.
1595
              BSIZE must be at least p_height * (p_width + 7) / 8.
1596
1597
              Data areas in buffer.b[] corresponding to non-visible areas of a
1598
              virtual screen will contain zeroes, i.e. cleared bits.
1599
              Note that the most significant bit in a byte is displayed to the left
1600
1601
              WARNING: At the moment this request is extremely slow and the computer
              seems to hang up while this request is served.
1602
1603
1604
1605
              6.33
                      Alter the Spray Mask
1606
                       ------
1607
1608
              This request changes the 32 times 32 pixels pattern used by the spray
1609
              escape sequence (see wh_escapes.doc).
1610
              The request is:
1611
1612
                      Spraymask(fd, bp)
```

```
1613
                        int
                                        fd:
1614
                        struct sprayst *bp;
1615
1616
                'fd' is the file descriptor for the window and the sprayst structure
1617
                looks like:
1618
1619
                       typedef unsigned long
                                                dword:
1620
1621
                       struct sprayst
1622
                       {
1623
                                dword
                                       sp_mask[8*sizeof(dword)];
1624
                       };
1625
1626
               where 'sp_mask[]' contains the bit pattern for the spray mask.
               Note that the most significant bit in a "dword" is displayed to the
1627
1628
               left on the screen.
1629
1630
1631
               7.
                       Other I/O Control Commands
1632
                       1633
1634
               This is a list of I/O control requests which are identical or similar
1635
               to their counterparts in the tty device driver:
1636
1637
                       PFNKLD
                                       Load ABC99 function keys. The file descriptor
1638
                                       can be both the one for a window and the one
                                       obtained when the window handler was
1639
1640
                                       activated.
1641
                       PFNKRD
                                       Read ABC99 function keys. The file descriptor
                                       can be both the one for a window and the one
1642
1643
                                       obtained when the window handler was
1644
                                       activated.
                                       Write data to the ABC99 keyboard. The file
1645
                       PTOKBD
1646
                                       descriptor must be the one obtained when the
                                       window handler was activated.
1647
                                       Fetch the basic parameters for the terminal
1648
                       TIOCGETP
                                       (v7).
1649
1650
                       TIOCSETP
                                       Flush and then set the basic parameters (v7).
1651
                       TIOCSETN
                                       Set the basic parameters (no flush) (v7).
                                       Set "exclusive-use" mode (v7).
1652
                       TIOCEXCL
                                       Turn off "exclusive-use" mode (v7).
1653
                       TIOCNXCL
1654
                       TIOCFLUSH
                                       Flush input and output queues (v7).
1655
                       TIOCSETC
                                       Set the special characters (v7).
1656
                                       Get the special characters (v7).
                       TIOCGETC
                                       Check if any character on input (v7).
1657
                       FIORDCHK
                                       Wait for output to drain, then flush the input
1658
                       TCSETAF
1659
                                       queues, and set the parameters for the
1660
                                       terminal.
1661
                       TCSETAW
                                       As above, but do not flush the input queues.
1662
                       TCSETA
                                       Set the parameters for the terminal.
1663
                       TCGETA
                                       Get the parameters for the terminal.
1664
                       TCFLSH
                                       Flush the input, output, or both the input and
1665
                                       output queues.
1666
1667
               It should be noted that the set up of the ABC99 function keys is
1668
               common for all windows. Hence the PFNKLD and PFNKRD requests should
1669
               be used carefully.
1670
1671
1672
               8.
                       Window Groups
1673
                       -----
```

1677 1678

1679 1680 1681

1682 1683

1684 1685

1686 1687 1688

1689 1690 1691

1692 1693

1694 1695 1696

1697 1698

1699 1700 1701

1702 1703

1704 1705

1706 1707

1708 1709 1710

1711 1712 1713

1718

1728 1729

1730 1731 1732

1733 1734 1735

1736

All windows belonging to the same process group and with the WGROUP flag set, belongs to a window group.

The parent window in a group is the first window in a process group created with the WGROUP flag set.

A child window is a window which is not a parent and which has the WGROUP flag set (i.e. the remaining windows in a group). If the parent disappears (i.e. is closed), the children looses their group connection.

It is guaranteed that all windows in one window group always are on consecutive levels.

9. Some Notes about the Storage of the Text Contents of a Virtual Screen

------

text contents and regards text as graphics:

If the SAVETEXT flag for a window is set, the window handler will internally store the text contents of the virtual screen and automatically update the window when necessary. There are two cases when the window handler stops remembering the

- If the escape sequence ESC: (n) H is sent to the window or
- ii) If the font is changed using the Select Character Set escape sequence.

There exists two possibilities to force the handler to start remembering the text contents again:

- Send the Reset to Initial State escape sequence (ESC c) to the window or
- Send the ESC: J escape sequence to the window when the current font is the same as the default font for the window.

Method i) has some side effects, so method ii) is to be prefered.

10. Functions Automatically Supported by the Window Handler 

The handler automatically moves a pointer around the screen when the mouse is moved.

If the pointer points to a region marked by the Winicon() request, the area is inverted if the I\_INVERT flag is set and if the left button on the mouse is pressed, the specified code sequence is sent to the appropriate process.

If the pointer points to a marked area in the lower right corner of a window border and the left button on the mouse is pressed, the size of the window can be changed by moving the mouse around. The operation is suspended when the left mouse button is released. If the window is a parent of a window group, the children will also be moved if appropriate.

To move a window (including the virtual screen) around, put the pointer on the mark at the upper right corner of the border, press the left button on the mouse and move the window by moving the mouse. To stop the operation, just release the button. If the moved window is a parent of a window group, the children will also be moved if appropriate.

 To change the part of the virtual screen which is visible in the window put the pointer on one of the four scroll arrows and press the left button on the mouse. This will cause the window to scroll one row or column in the direction indicated by the arrow. An alternative is to put the mouse pointer on the horizontal or the vertical visible indicator, press the left button, and drag the indicator to the desired location. The window is scrolled when the left button is released.

If the pointer is put on the mark at the upper left corner of the border and the left button on the mouse is pressed, a signal (if specified) will be sent to all processes in the window.

To copy a region (a rectangle) of text from one window to another, put the pointer at the upper left character of the rectangle, press the middle button on the mouse and a rectangle can now be made by moving the pointer to the lower right character and releasing the button. The marked region is now indicated by four lines surrounding it. To abort the operation, press any button, except the middle one, otherwise move the pointer to the destination window and press the middle button once more, causing the marked region to be copied. Note that this operation will also work with programs not knowing about the windows, since the text contents of all the windows are stored by the window handler.

To make a window the top level window, put the mouse pointer on the window and press the right mouse button. If the window already is the top level window, the window is moved to the bottom instead. If the pointer is pointing to the background or a special window, the top level window is put at the bottom.

If the window to be moved to the top or the bottom belongs to a window group, the whole group is moved without affecting the relative levels inside the group.