This file contains descriptions of the window utility commands.

1. Wopen

This command creates a new window with the status of a terminal and executes the command given as argument in it (if no command is specified, a shell is executed).
The syntax is:

```
        wopen [-notbwz] [-c <n>] [-r <n>] [-h <n>] [-w <n>] [-x <n>]
              [-y <n>] [-f <c>] [-s <n>] [-e <n>] [<command>]
```

Explanation of the options:

```
  b - Black window.
  w - White window (this is the default).
  n - No window border.
  o - Single (one) line window border.
  t - Double (two) lines window border (this is the default).
  z - Zoom box shall be present in the border.
  c - Number of character columns in the window (default 80).
  r - Number of character rows in the window (default 24).
  h - height of window in pixels.
  w - width of window in pixels.
  x - x coordinate of the lower left corner of the window (default 24
      in portrait mode and 152 in landscape mode).
  y - y coordinate of the lower left corner of the window (default 344
      in portrait mode and 216 in landscape mode).
  f - The default font to be used (default 'A').
  s - Signal to be used to signal that the window has moved, etc.
      (default 0).
  e - Signal to be sent when the close box is used. If not zero, a
      close (exit) box will be present in the border (default 0).
```

2. Whead

This command inserts a header in a window.
The syntax is:

```
        whead [-i] [-t] [<header>]
```

Explanation of the options:

```
  i - Invert the header.
  t - Invert the top header.
```

If no header is given, the present header will be removed.

3. Wicon

This command sets up an icon in a window.
The syntax is:

```
        wicon [-prielmqszt] [-x <n>] [-y <n>] [-w <n>] [-h <n>]
              [<sequence>]
```

Explanation of the options:

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

r - Send icon sequence when left mouse button is released.
i - Invert the icon when the mouse pointer points to it.
e - Send the icon sequence when we enter the icon area.
l - Send the icon sequence when we leave the icon area.
m - Remove the icon after the icon sequence has been sent.
q - Only send the icon sequence if there is a pending read request on the window.
s - Check if option e or l is fulfilled upon set up.
z - Only send the icon sequence if it is the level zero window.
t - The coordinates and sizes are supposed to be given in character box units.
x - The x coordinate of the lower left corner of the icon (default 0).
y - The y coordinate of the lower left corner of the icon (default 0).
w - The width of the icon (default 100).
h - The height of the icon (default 100).

<sequence> is the icon sequence to be sent when the icon is chosen.

4. Rmicons

This command removes all icons in a window.
The syntax is:

```
rmicons
```

5. Wzoom

This command sets up a zoom list for a window.
The syntax is:

```
wzoom [<zoomlist>]
```

<zoomlist> is a string of capital letters indicating the fonts which the zoom list shall consist of. If no <zoomlist> is specified, any existing zoomlist is removed.

6. Wfont

This command changes the default font for a window.
The syntax is:

```
wfont [-x <n>] [-y <n>] [<font>]
```

Explanation of the options:
x - The x coordinate for the middle visible character (default 1).
y - The y coordinate for the middle visible character (default 1).
<font> is a single capital letter specifying the new font.
If no <font> is specified, the next font in the zoom list for the window is used instead.

7. Wtop

```
        This command moves a window to the top level.
        The syntax is:

                wtop


        8.      Wbg
                ===

        This command reads the file specified as argument and uses the data to
        set up a new background pattern for the window handler. It supposes
        file descriptor 3 to be the window handler "super" channel.
        The syntax is:

                wbg [-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.


        9.      Wmsk
                ====

        This command reads the file specified as argument and uses the data to
        set up new mouse substitute keys for the window handler. It supposes
        file descriptor 3 to be the window handler "super" channel.
        The syntax is:

                wmsk [-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.


        10.     Wmp
                ===

        This command reads the file specified as argument and uses the data to
        set up a new global mouse pointer for the window handler. It supposes
        file descriptor 3 to be the window handler "super" channel.
        The syntax is:

                wmp [-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.


        11.     Widtp
                =====

        This command reads the file specified as argument and uses the data to
        set up new initial driver and terminal parameters for the window
        handler. It supposes file descriptor 3 to be the window handler
        "super" channel.
        The syntax is:

                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 <n>] [-y <n>] [-w <n>] [-h <n>] [-c <n>]
                [-o <file>]
```

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 <n>] [-y <n>] [-u <n>] [-v <n>] [-w <n>]
               [-h <n>]
```

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).

  w - Width of the window.
  h - Height of the window.

15. Whelp

This command changes the sequence sent when the help box is used.
The syntax is:

```
        whelp [(sequence)]
```

No sequence will be sent if (sequence) is not given.

1985-07-29

ABC1600 WINDOW HANDLER ESCAPE SEQUENCES

This documentation briefly describes all the escape sequences implemented in the window handler. There are two types: VT100 and/or Facit Twist compatible sequences and sequences private to the ABC1600.
The sequences are, if possible, compatible with the ones used in the ABC1600 terminal emulator (the console).

1. VT100 and Facit Twist Compatible Escape Sequences

1.1 Cursor Up

ESC[⟨Pn⟩A

Moves the text cursor ⟨Pn⟩ lines up. The cursor stops at the top margin. If ⟨Pn⟩ is zero or not present, the cursor is moved one line upwards.

1.2 Cursor Down

ESC[⟨Pn⟩B

Moves the text cursor ⟨Pn⟩ lines down. The cursor stops at the bottom margin. If ⟨Pn⟩ is zero or not present, the cursor is moved one line down.

1.3 Cursor Forward

ESC[⟨Pn⟩C

Moves the text cursor ⟨Pn⟩ positions to the right. The cursor stops at the right margin. If ⟨Pn⟩ is zero or not present, the cursor is moved one position to the right.

1.4 Cursor Backward

ESC[⟨Pn⟩D

Moves the text cursor ⟨Pn⟩ positions to the left. The cursor stops at the left margin. If ⟨Pn⟩ is zero or not present, the cursor is moved one position to the left.

1.5 Cursor Position

ESC[⟨Pn⟩;⟨Pn⟩H   or   ESC[⟨Pn⟩;⟨Pn⟩f

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

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
0	Erase from and including the current text cursor position to the end of the scrolling region (default).
1	Erase from the start of the scrolling region to and including the current text cursor position.
2	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
0	Erase from and including the current text cursor position to the end of the line (default).
1	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 performed.

1.10 Next Line

ESC E

Moves the text cursor to the first position on the next line downward. If the cursor is at the bottom margin, a scroll up is performed.

1.11 Reverse Index

ESC M

Moves the text cursor one line upward without changing the column position. If the cursor is at the top margin, a scroll down is performed.

1.12 Save Cursor

ESC 7

Saves the current text cursor position, graphic cursor position, graphic origin, character attributes, and character font.

1.13 Restore Cursor

ESC 8

Restores all things saved by the Save Cursor sequence to the state when the Save Cursor sequence was last used. If no Save Cursor sequence has been sent to the window the text cursor, graphic cursor, and graphic origin are set to their home positions.

1.14 Reset to Initial State

ESC c

On a VT100 terminal this sequence resets it to its initial state. To simulate this in a window, the following things are performed when this sequence is received:

- The text cursor is put at its home position.
- The graphic cursor is put at its home position.
- The text cursor apperance is set to the default.
- The Set Mode - Reset Mode flags are set to their default values.
- The character attributes are set to their default values.
- The top and bottom margin of the scrolling region are set to the the top and bottom line of the virtual screen.
- The graphic origin is set to the lower left corner of the virtual screen.
- Tab stops are set to the default.
- The graphic pattern tables are set to their default values.

- The current font is set to the default font for the window.
- The whole virtual screen is cleared.
- All the LED's on the keyboard are turned off.

1.15 Tabulation Backward

ESC[Z

Moves the text cursor left to the next tab stop. The cursor stops at the left margin.

1.16 Horizontal Tabulation Set

ESC H

Set a horizontal tabulation stop at the current text cursor position.

1.17 Tabulation Clear

ESC[(Ps)g

If (Ps) is 0 the horizontal tab stop at the current text cursor position is cleared (default).
If (Ps) is 3 all horizontal tab stops are cleared.

1.18 Character Attributes

ESC[(Ps);(Ps);....;(Ps)m

Set or reset character attributes according to the parameter(s):

Parameter	Meaning
0	Attributes off.
1	Bold or increased intensity. On the ABC1600 this has the same effect as the reverse character attribute.
4	Underscore.
5	Blink. On the ABC1600 this has the same effect as the reverse character attribute.
7	Reverse.

1.19 Device Status Report

ESC[(Ps);(Ps);...;(Ps)n

Request to get a report of the specified status. The status is determined by the parameter(s).

Parameter	Meaning
6	Report the text cursor position. The report sequence is ESC[(Pn);(Pn)R where the first parameter

	specifies the line and the second the column.
?1	Report Portrait/Landscape screen mode. This is compatible with the Facit Twist terminal. The report sequence is ESC[?Pn for portrait mode and ESC[?Ln for landscape mode.

1.20   Load LEDs

ESC[<Ps>;<Ps>;...;<Ps>q

Loads the eight programmable LEDs on the keyboard according to the parameter(s).

Parameter	Meaning
0	Clear LEDs 1 through 8.
1	Light LED 1.
2	Light LED 2.
3	Light LED 3.
4	Light LED 4.
5	Light LED 5.
6	Light LED 6.
7	Light LED 7.
8	Light LED 8.

The default value of the parameter is 0.
Note that the status of the keyboard LEDs always reflects the LED status for the top level window.

1.21   Set Mode

ESC[<Ps>;<Ps>;...;<Ps>h

Sets the modes specified by the parameter(s). The different modes are:

Parameter	Meaning
20	Line feed new line mode. When set causes the LF key to imply movement to the first position of the following line and causes the RETURN key to send both CR and LF.
?5	Screen mode. When set the window is inverted.
?6	Origin mode. When set the home position for the text cursor is at the upper-left position of the scrolling region.
?7	Auto wrap mode. When set, the text cursor will advance to the next line when it reaches the right margin.
?32	Page mode, i.e. the window does not scroll. This is compatible with the Facit Twist terminal.
?33	Underline cursor. This is compatible with the Facit Twist terminal.
?34	Blinking cursor. This is compatible with the Facit Twist terminal.
?35	Cursor off. This is compatible with the Facit Twist terminal.

1.22 Reset Mode

```
ESC[<Ps>;<Ps>;...;<Ps>l
```

Resets the modes specified by the parameter(s). The different modes are:

Parameter	Meaning
20	Line feed new line mode. When reset causes the LF key to imply only vertical movement of the text cursor and the RETURN key to send the single code CR.
?5	Screen mode. When reset the window is not inverted.
?6	Origin mode. When reset the text cursor home position is at the upper-left position of the virtual screen.
?7	Auto wrap mode. When reset, the text cursor will not advance to the next line when it reaches the right margin.
?32	Scroll mode. This is compatible with the Facit Twist terminal.
?33	Reverse block cursor. This is compatible with the Facit Twist terminal.
?34	Non-blinking cursor. This is compatible with the Facit Twist terminal.
?35	Cursor on. This is compatible with the Facit Twist terminal.

1.23 Select Character Set

```
ESC(A     or     ESC)A
ESC(B     or     ESC)B
       .
       .
       .
ESC(Z     or     ESC)Z
```

Selects the desired font. When changing between fonts of different sizes, the fonts will be aligned so that the base lines of the fonts will be the same.
Note that when the font is changed for a window, the saving of the text contents of the window will be lost.

2. ABC1600 Private Escape Sequences

2.1 Draw Line

```
ESC:<x>;<y>;<pno>;<cno>d
```

Draws a line from the current graphic cursor position to <x>,<y>, using the pattern specified by <pno>. If the colour number <cno> is '1' a normal line is drawn and if it is '0' or not specified the line is the inverse of that obtained with the colour number '1'. If <pno> is not specified, a continous line is drawn.
The graphic cursor position is updated to <x>,<y>.

2.2   Draw Inverted Line

```
ESC:<x>;<y>i
```

Draws a line from the current graphic cursor position to <x>,<y> by inverting the corresponding pixels. The line can be removed by drawing an inverted line a second time.
The graphic cursor position is updated to <x>,<y>.

2.3   Move Graphic Cursor

```
ESC:<x>;<y>m
```

Positions the graphic cursor at <x>,<y>.

2.4   Draw Point

```
ESC:<x>;<y>;<op>;<cno>p
```

Changes or reads the pixel at <x>,<y>. <op> determines the operation:

```
  If <op> is 0 or not specified, set the pixel.
  If <op> is 1, clear the pixel.
  If <op> is 2, complement the pixel.
  If <op> is 10, the colour of the pixel at <x>,<y> is reported:

        ESC:<x>;<y>;11;<cno>p          <cno> is '1' if the pixel is
                                       set, otherwise '0'.
        ESC:<x>;<y>;11p                The specified pixel is outside
                                       the virtual screen.
```

The graphic cursor position is updated to <x>,<y> if <op> is 0, 1, or 2.
Note that <cno> is not used for <op> equal to 0, 1, 2, or 10 and may be left out.

2.5   Draw Arc

```
ESC:<x>;<y>;<len>;<pno>;<cno>a
```

Draws a circle arc with the origin at <x>,<y> from the current graphic cursor position counter-clockwise with length <len> using the pattern <pno>. If <pno> is not specified, a continous arc is drawn.
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.
If the colour number <cno> is '1', a normal arc is drawn and if it is '0' or not specified the arc is the inverse of that obtained with the colour number '1'.
The graphic cursor position is updated to the last drawn pixel in the arc.

2.6   Draw Inverted Arc

ESC:<x>;<y>;<len>I

Draws a circle arc, with the origin at <x>,<y>, from the current graphic cursor position counter-clockwise with length <len> by inverting the corresponding pixels.
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.

2.7 Fill Area

ESC:<x>;<y>;<pno>;<cno>f

Fills a rectangle with the pattern <pno>. If <pno> is not specified, all pixels in the rectangle are set.
The rectangle has one of its corners at <x>,<y> and the opposite corner at the current graphic cursor position.
If the colour number <cno> is '1', a normal fill is done and if it is '0' or not specified, the rectangle is the inverse of that obtained with colour number '1'.
The graphic cursor position is updated to <x>,<y>.

2.8 Draw Filled Circle

ESC:<x>;<y>;<rad>;<pno>;<cno>c

Draws a filled circle with origin at <x>,<y> and with radius <rad> using the pattern <pno>. If <pno> is not specified, all pixels in the circle are set.
If the colour number <cno> is '1', a normal fill is done and if it is '0' or not specified, the circle is the inverse of that obtained with colour number '1'.
The graphic cursor position is updated to <x>,<y>.

2.9 Paint Area

ESC:<x>;<y>;<pno>;<cno>F

Paints an area with the pattern <pno>. The area to be painted should be limited by continous lines (curves) generated by previous line, dot, circle, fill, paint, etc. operations.
<x>,<y> specifies the starting point for the paint and should be within the area. If the pixel at <x>,<y> is cleared, the limits of 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 "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 '0' or not specified, the paint is the inverse of that obtained with colour number '1'.
The graphic cursor position is updated to <x>,<y>.
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.

2.10 Move Area

```
ESC:<xsrc>;<ysrc>;<xdest>;<ydest>;<width>;<height>;<op>r
```

Moves (actually copies) the rectangular area with lower left corner at <xsrc>,<ysrc> to <xdest>,<ydest>. The area has width <width> and 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.
The graphic cursor position is not updated.
Note that only those areas where both the source and destination areas are visible are moved.

2.11 Define Pattern

```
ESC:<pno>;<hmask>;<vmask>;<shift>;<op>R
```

Redefines the pattern <pno> as specified. The pattern is defined for portrait mode and will be tilted 90 degrees when used in landscape mode.
<hmask> defines a 16 bit horizontal mask used repeatedly on a scan line during fill or when drawing lines or arcs.
<vmask> defines a 16 bit vertical mask where each bit determines the operation on the corresponding scan line. If a bit is set <hmask> is used to fill the scan line, otherwise <op> determines the operation:

<op> = 0	Clear the line, rotate <hmask> the number of bits given by <shift>.
<op> = 1	Set the line, rotate <hmask> the number of bits given by <shift>.
<op> = 2	Use <hmask> but complemented, rotate <hmask> the number of bits given by <shift>.
<op> = 3	Leave line as it is, rotate <hmask> the number of bits given by <shift>.
<op> = 4	Clear the line, no rotate.
<op> = 5	Set the line, no rotate.
<op> = 6	Use <hmask> but complemented, no rotate.
<op> = 7	Leave line as it is, no rotate.

<pno> can be in the range 1 - 15. Pattern number zero can not be redefined.
<shift> can be in the range 0 - 15.
Only <hmask> is used by the draw line and draw arc escape sequences.

2.12 Set Text Cursor

```
ESC:<sel>H
```

The text cursor is positioned at the position of the graphic cursor according to <sel>:

<sel> = 0	The upper left corner of the font box is placed at the graphic cursor.
<sel> = 1	The lower left corner of the font box is placed at the graphic cursor.

(sel) = 2   The left edge of the base line for the font box is placed at the graphic cursor.

Note that when this escape sequence is sent to a window, the saving of the text contents of the window will be lost.

2.13 Mouse Report

`ESC:<sel>M`

This escape sequence is used to get a report of the current mouse pointer position. The report is, depending on <sel>, only sent when the mouse pointer or the mouse buttons have changed.

(sel) = 7   The report is sent immediately if the mouse has changed since the last report. Otherwise the report is sent as soon as the mouse changes. A change is either a mouse movement or a status change of a mouse button.
The report sequence is:

```
ESC:<x>;<y>;<buttons>P
```

where <x> and <y> is the position of the mouse pointer. If the mouse pointer is outside the virtual screen, the reported position will be at the virtual screen border.
<buttons> is '1' if the left button is pressed, '2' if the middle button is pressed, '3' if both the left and middle buttons are pressed, and '0' if no button is pressed.

(sel) = 8   Identical to <sel> = 7, except that reports are only sent when the left or middle buttons changes.

Note that mouse reports are only sent to the top level window.

2.14 Device Status Report

`ESC:<sel>n`

Reports the status of different devices, determined by <sel>:

(sel) = 1   Reports the graphic cursor position. The report sequence is:

```
ESC:<x>;<y>R
```

where <x>,<y> is the current graphic cursor position.

(sel) = 2   Reports the mouse position and button status. This is identical to the Mouse Report escape sequence with <sel> = 7 (ESC:7M), except that the report is sent immediately.
Note that reports are only sent to the top level window.

(sel) = 3   Reports the size of the virtual screen and the current font. The report sequence is:

```
ESC:<vsx>;<vsy>;<fsx>;<fsy>;<bl>;<fno>W
```

⟨vsx⟩ and ⟨vsy⟩ are the x and y pixel sizes, respectively, of the virtual screen, ⟨fsx⟩ and ⟨fsy⟩ are the x and y pixel sizes of the current font box, ⟨bl⟩ is the base line for the font box, and ⟨fno⟩ is the ASCII code for the name of the current font.

2.15 Set Graphic Origin

```
ESC:⟨x⟩;⟨y⟩O
```

Sets the graphic origin to ⟨x⟩,⟨y⟩. The graphic cursor position is set to 0,0.
All coordinates given by the graphic escape sequences are relative to the graphic origin.
Note that the mouse position is always reported relative to the lower left corner of the virtual screen.

2.16 Clear All

```
ESC:J
```

Clear window and home cursors, etc. as follows:

- The text cursor is set to 1,1.
- The graphic cursor is set to 0,0.
- The graphic origin is set to 0,0.
- The scroll region is reset to the whole virtual screen.
- If the current character font is the same as the default font for the window, the text contents of the window will be started to be remembered again.
- The whole virtual screen is cleared.

2.17 Load Key LEDs

```
ESC:⟨sel⟩;⟨sel⟩;...;⟨sel⟩q
```

Loads the LEDs on the INS and ALT keys according to the parameter(s).

⟨sel⟩ = 0	Clear both the LEDs.
⟨sel⟩ = 1	Light the INS key LED.
⟨sel⟩ = 2	Light the ALT key LED.

If no parameter is specified, the LEDs are cleared.
Note that the status of the keyboard LEDs always reflects the LED status for the top level window.

2.18 Private Set Mode

```
ESC:⟨sel⟩;⟨sel⟩;...;⟨sel⟩h
```

Sets the ABC1600 private modes specified by the parameter(s). The different modes are:

⟨sel⟩ = 2	Phased pattern mode. When set, the patterns obtained when using the fill area, draw filled circle, paint area, and spray escape sequences will be phased.

2.19 Private Reset Mode

ESC:⟨sel⟩;⟨sel⟩;...;⟨sel⟩l

Resets the ABC1600 private modes specified by the parameter(s). The different modes are:

⟨sel⟩ = 2	Non-phased pattern mode. When reset, the patterns obtained when using the fill area, draw circle, paint area, and spray escape sequences will not be phased.

2.20 Spray

ESC:⟨x⟩;⟨y⟩;⟨pno⟩;⟨op⟩s

This escape sequence manipulates the pixels which are set both in the spray mask and in the pattern specified by ⟨pno⟩, according to the operation ⟨op⟩.
⟨x⟩,⟨y⟩ is the lower left corner of where to put the 32x32 pixels spray mask.
If ⟨pno⟩ is not specified, '0' is used and if ⟨op⟩ is not specified, '0' is used.
The following operations can be performed:

⟨op⟩ = 0	All pixels which are set both in the spray mask and in the pattern are set and the remaining pixels are cleared (replace).
⟨op⟩ = 1	All pixels which are set both in the spray mask and in the pattern are set. The remaining pixels are left unaffected (set).
⟨op⟩ = 2	All pixels which are set both in the spray mask and in the pattern are cleared. The remaining pixels are left unaffected (reset).
⟨op⟩ = 3	All pixels which are set both in the spray mask and in the pattern are complemented. The remaining pixels are left unaffected (complement).

The spray mask for a window can be altered by a request to the window handler.
For most applications of this escape sequence, the window must probably be set to phased pattern mode in order to give a meaningful result.
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 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 refered to are the ones used to communicate with the window handler and can be found in the documentation for the handler):

window	Data for a window (the winstruc structure).
icon	Data for an icon (the winicon structure).
string	A string to be used in both landscape and portrait mode.
pstring	A string to be used in portrait mode only.
lstring	A string to be used in landscape mode only.
pointer	Data for the layout of a mouse pointer (the npstruc structure).
header	Data for a window header (the headstruc structure).
environ	Environment strings used to modify the environment in both portrait and landscape screen mode.
penviron	Environment strings to be used in portrait mode only.
lenviron	Environment strings to be used in landscape mode only.
directory	A directory pathname.
command	A command (the file name and the arguments).
flags	Window flags data (the flgstruc structure).
zoomlist	Zoom list data (the zoomlst structure).
substitute	Mouse substitute keys (the substit structure).
background	Data for a background pattern (the chbgstruc 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 '0x' or '0X' 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.

3.1.1 Window

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.

Keyword	Description
pxor#	(wp_xorig) The x coordinate in portrait mode of the lower left corner of the virtual screen.
lxor#	(wl_xorig) The x coordinate in landscape mode of the lower left corner of the virtual screen.
pyor#	(wp_yorig) The y coordinate in portrait mode of the lower left corner of the virtual screen.
lyor#	(wl_yorig) The y coordinate in landscape mode of the lower left corner of the virtual screen.
pxsi#	(wp_xsize) The horizontal size in portrait mode of the virtual screen.
lxsi#	(wl_xsize) The horizontal size in landscape mode of the virtual screen.
pysi#	(wp_ysize) The vertical size in portrait mode of the virtual screen.
lysi#	(wl_ysize) The vertical size in landscape mode of the virtual screen.
pvxo#	(wp_vxorig) The x coordinate in portrait mode of the lower left corner of the window relative to the lower left corner of the virtual screen.
lvxo#	(wl_vxorig) The x coordinate in landscape mode of the lower left corner of the window.
pvyo#	(wp_vyorig) The y coordinate in portrait mode of the lower left corner of the window.
lvyo#	(wl_vyorig) The y coordinate in landscape mode of the lower left corner of the window.
pvxs#	(wp_vxsize) The horizontal size in portrait mode of the window.
lvxs#	(wl_vxsize) The horizontal size in landscape mode of the window.
pvys#	(wp_vysize) The vertical size in portrait mode of the window.
lvys#	(wl_vysize) The vertical size in landscape mode of the window.
colr#	(w_color) Background colour in the window. 0 = Black, 1 = White.
bord#	(w_border) The type of the window border. The different types are (N = No border, S = Single line border, D = Double lines border):

Border type	Left side	Right side	Upper side	Lower side

0	N	N	N	N
1	S	S	S	S
2	D	D	D	D
3	D	S	S	S
4	S	D	S	S
5	S	S	D	S
6	S	S	S	D
7	D	D	S	S
8	D	S	D	S
9	D	S	S	D
10	S	D	D	S
11	S	D	S	D
12	S	S	D	D
13	D	D	D	S
14	D	D	S	D
15	D	S	D	D
16	S	D	D	D

pfnt#	(wp_font) The initial font in portrait mode (ASCII code, i.e. font A is 65).
lfnt#	(wl_font) The initial font in landscape mode.
usrb#	(w_uboxes) The maximal number of user defined boxes that can be set up in the left side of the border.
tsig#	(w_tsig) The signal used to signal that the window has moved to the top level. The window shell always sets this one to zero for menu windows.
nsig#	(w_ntsig) The signal used to signal that the window has moved from the top level. The window shell always sets this one to zero for menu windows.
rsig#	(w_rsig) The signal used to signal a window that it has to redraw itself. The window shell sets this one to zero for menu windows if the 'stxt' flag is present. If the 'stxt' flag is not set, the window shell sets this signal to an appropriate value.
csig#	(w_csig) The signal to be sent to processes in a window when the close box is used.
hscr	(BX_HSCR) The scroll left and right boxes shall be present in the border.
vscr	(BX_VSCR) The scroll up and down boxes shall be present in the border.
cbox	(BX_CLOS) The close box shall be present in the border. The window shell clears this flag for menu windows.
sbox	(BX_SIZE) The size box shall be present in the border.
mbox	(BX_MOVE) The move box shall be present in the border.
zbox	(BX_ZOOM) The zoom box shall be present in the border.
avis	(BX_AVIS) Scroll left/right and up/down are only visible if the whole virtual screen is not visible.
bbox	(BX_BLOW) The blow up box shall be present in the border.
hbox	(BX_HELP) The help box shall be present in the border.
pmod	(PMODE) Portrait mode coordinates given.
lmod	(LMODE) Landscape mode coordinates given.
stxt	(SAVETEXT) Save the text contents of the virtual screen.
sbmp	(SAVEBITMAP) Save the bitmap contents of the virtual screen (future use).
lock	(LOCK) Lock the window on the top level.
novr	(NOOVER) The window must not be overlapped by another window.
ncur	(NOCURSOR) Text cursor not visible.
nmov	(NOMOVE) The window must not be moved or change size.

alls	(ALLSCR) The window must be the whole virtual screen.
spec	(SPECIAL) Special window.
kscr	(KEYSCROLL) Make sure that the text cursor is visible in the window everytime a key is pressed.
wscr	(WRITSCROLL) Make sure that the text cursor is visible in the window everytime something has been written to the window.
amsp	(ALTMPNT) Allocate space to store a private mouse pointer for the window.
rltv	(RELATIVE) Add the window relative to the parent window.
ncpi	(NOCPIN) Prevents text from being copied into this window.
ncpo	(NOCPOUT) Prevents text from being copied from this window.
text	(TXTSIZE) The size of the virtual screen, the window, and the origin of the window are supposed to be given in term of characters instead of pixels.
wgrp	(WGROUP) This window shall belong to a window group.
rulc	(REL_ULC) This window shall follow its parent window relative the upper left corner of the parent (not meaningful if the window is not a child window).
rurc	(REL_URC) This window shall follow its parent window relative the upper right corner of the parent (not meaningful if the window is not a child window).
rllc	(REL_LLC) This window shall follow its parent window relative the lower left corner of the parent (not meaningful if the window is not a child window).
rlrc	(REL_LRC) This window shall follow its parent window relative the lower right corner of the parent (not meaningful if the window is not a child window).

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.

Keyword	Description
pxor#	(ip_xorig) The x coordinate in portrait mode of the lower left corner of the icon.
lxor#	(il_xorig) The x coordinate in landscape mode of the lower left corner of the icon.
pyor#	(ip_yorig) The y coordinate in portrait mode of the lower left corner of the icon.
lyor#	(il_yorig) The y coordinate in landscape mode of the lower left corner of the icon.
pxsi#	(ip_xsize) The horizontal size in portrait mode of the

	icon.
lxsi#	(il_xsize) The horizontal size in landscape mode of the icon.
pysi#	(ip_ysize) The vertical size in portrait mode of the icon.
lysi#	(il_ysize) The vertical size in landscape mode of the icon.
cseq=	(i_cmdseq[]) Character sequence to be sent by the icon.
pmod	(I_PMODE) Portrait mode coordinates given.
lmod	(I_LMODE) Landscape mode coordinates given.
pres	(I_PRESS) Send sequence when left button is pressed.
rlse	(I_RELEASE) Send sequence when left button is released.
inve	(I_INVERT) Invert icon when we are pointing to it.
entr	(I_ENTER) Send sequence when we are moving into the icon area.
leav	(I_LEAVE) Send sequence when we are leaving the icon area.
rmov	(I_REMOVE) Remove the icon when a sequence has been sent.
rqst	(I_RQST) Only send the sequence if there is a pending read request to the window.
schk	(I_SETCHK) Check if 'entr' or 'leav' is fulfilled when setting up the icon.
lzer	(I_LZERO) The sequence is sent only if the window is at the top level.
text	(I_TEXT) The icon coordinates are supposed to be in character units.

The following example puts the icon in the lower left corner of a virtual screen (only portrait mode coordinates are given):

```
icon17:pxor#0:pyor#0:pxsi#80:pysi#50:cseq=\200:\
      :pmod:pres:rlse:inve:rqst
```

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

3.1.3   String, pstring, and lstring

To set up the string

```
I like
WINDOWS!
```

using string, pstring, or lstring, looks like:

```
string36:I like\nWINDOWS!
pstring12:I like\nWINDOWS!
lstring19:I like\nWINDOWS!
```

3.1.4   Pointer

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).

Keyword	Description
xsiz#	(np_xsize) The with of the mouse pointer.
ysiz#	(np_ysize) The height of the mouse pointer.
xpnt#	(np_xpnt) Pointing part of the mouse pointer, x coordinate.
ypnt#	(np_ypnt) Pointing part of the mouse pointer, y coordinate.
andm#	(np_and[]) A series of 16 AND masks used to construct the mouse pointer. The different elements are separated by commas.
orma#	(np_or[]) A series of 16 OR masks used to construct the mouse pointer. The different elements are separated by commas.

The following is an example of a black hair cross mouse pointer:

```
  pointer7:xsiz#31:ysiz#31:xpnt#15:ypnt#15:\
          :andm#0xfffeffff,0xfffeffff,0xfffeffff,0xfffeffff,\
                0xfffeffff,0xfffeffff,0xfffeffff,0xfffeffff,\
                0xfffeffff,0xfffeffff,0xfffeffff,0xfffeffff,\
                0xfffeffff,0xfffeffff,0xfffeffff,0x00000001,\
                0xfffeffff,0xfffeffff,0xfffeffff,0xfffeffff,\
                0xfffeffff,0xfffeffff,0xfffeffff,0xfffeffff,\
                0xfffeffff,0xfffeffff,0xfffeffff,0xfffeffff,\
                0xfffeffff,0xfffeffff,0xfffeffff,0xffffffff\
          :orma#0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

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

3.1.5 Header

The header data item gives the data for a window header. Every keyword has a corresponding member or flag in the headstruc structure (see the documentation for the window handler).

Keyword	Description
head=	(h_hdr[]) The header string.
invh	(H_INVHD) Invert the window header.
invt	(H_INVT) Invert the top window header.

The following is an example of the header ' MY PROGRAM ':

```
  header17:head= MY PROGRAM :invt
```

3.1.6 Directory

To specify the directory pathname /usr/sven/bin, use the line:

```
  directory4:/usr/sven/bin
```

3.1.7 Environ, penviron, and lenviron

These data items specifies how the environment for the program

shall be modified before it is executed by wsh. If the specified environment variable already exist, the old one is replaced. Otherwise the environment variable is added to the environment list.
To specify PATH to be '/usr/mydir/bin' and TERM to be 'vt100', use the line:

```
  environ3:PATH=/usr/mydir/bin,TERM=vt100
```

By modifying the environment it is possible to tell programs, which uses termcap, the size of the virtual screen. If the size of the virtual screen is 132 columns times 33 lines, use:

```
  environ7:TERM=win,TERMCAP=w0|win|w:co#132:li#33:tc=abc1600w:
```

'abc1600w' is an entry in the termcap file which should be used for this purpose only.

The syntax for penviron and lenviron is equivalent.

3.1.8 Command

To specify the 'ls -l' command, use the line:

```
  command1:/bin/ls,ls,-l
```

'/bin/ls' is the file name, 'ls' is argument 0, '-l' is argument 1.

3.1.9 Flags

The flags data item gives the data for new window flags. Every keyword has a corresponding flag in the flgstruc structure (see the window handler documentation).

Keyword	Description
lock	(LOCK) See the description of the window data item.
novr	(NOOVER)
ncur	(NOCURSOR)
nmov	(NOMOVE)
alls	(ALLSCR)
kscr	(KEYSCROLL)
wscr	(WRITSCROLL)
ncpi	(NOCPIN)
ncpo	(NOCPOUT)
rulc	(REL_ULC)
rurc	(REL_URC)
rllc	(REL_LLC)
rlrc	(REL_LRC)

The following example can be used to set the LOCK flag for the window in the example in section 3.1.1:

```
  flags56:lock
```

3.1.10 Zoomlist

The zoomlist data item gives the data for a zoom list. Every keyword has a corresponding member or flag in the zoomlst structure (see the documentation for the window handler).

Keyword	Description
plst=	(zp_list[]) The set of fonts to be used in portrait mode.
llst=	(zl_list[]) The set of fonts to be used in landscape mode.
pmod	(Z_PMODE) Portrait mode list given.
lmod	(Z_LMODE) Landscape mode list given.

The following is an example of a zoom list (only data for portrait mode is given) which will make it possible to toggle between the window's default font and the font F:

```
zoomlist7:plst=F:pmod
```

3.1.11 Substitute

The substitute data item gives the data for a set of mouse substitute keys. Every keyword has a corresponding member in the substit structure (see the window handler documentation).

Keyword	Description
init#	(c_initflg) Flag indicating if the substitute keys are enabled or not after the set up (1 if enabled, 0 if not).
onof#	(c_keys[S_ONOFF]) Key used to toggle the substitute keys on or off.
mpup#	(c_keys[S_MPU]) Move mouse pointer up.
mpdo#	(c_keys[S_MPD]) Move mouse pointer down.
mple#	(c_keys[S_MPL]) Move mouse pointer left.
mpri#	(c_keys[S_MPR]) Move mouse pointer right.
mpul#	(c_keys[S_MPUL]) Move mouse pointer up - left.
mpur#	(c_keys[S_MPUR]) Move mouse pointer up - right.
mpdl#	(c_keys[S_MPDL]) Move mouse pointer down - left.
mpdr#	(c_keys[S_MPDR]) Move mouse pointer down - right.
lpup#	(c_keys[S_LMPU]) Move mouse pointer up a long step.
lpdo#	(c_keys[S_LMPD]) Move mouse pointer down a long step.
lple#	(c_keys[S_LMPL]) Move mouse pointer left a long step.
lpri#	(c_keys[S_LMPR]) Move mouse pointer right a long step.
lpul#	(c_keys[S_LMPUL]) Move mouse pointer up - left a long step.
lpur#	(c_keys[S_LMPUR]) Move mouse pointer up - right a long step.
lpdl#	(c_keys[S_LMPDL]) Move mouse pointer down - left a long step.
lpdr#	(c_keys[S_LMPDR]) Move mouse pointer down - right a long step.
pcmd#	(c_keys[S_PCMD]) Replacement for the left mouse button.
cwin#	(c_keys[S_CHWIN]) Replacemet for the right mouse button.
mtxt#	(c_keys[S_MCA]) Replacement for the middle mouse button.
step#	(c_step) Step length for a normal move of the mouse pointer.

lstp#	(c_lstep) Step length for a long move of the mouse pointer.

The following is an exemple of the set up of the mouse substitute keys:

```
  substitute1:init#0:onof#0xfe:mpup#0xa1:mpdo#0xa3:mple#0xac:\
              :mpri#0xa4:mpul#0xad:mpur#0xa5:mpdl#0xaf:mpdr#0xa7:
              :lpup#0xb1:lpdo#0xb3:lple#0xbc:lpri#0xb4:lpul#0xbd:\
              :lpur#0xb5:lpdl#0xbf:lpdr#0xb7:pcmd#0xcc:cwin#0xce:\
              :mtxt#0xcd:step#4:lstp#10
```

3.1.12 Background

The background data item gives the data for a background pattern. The keyword has a corresponding member in the chbgstruc structure (see the documentation for the window handler).

Keyword	Description
bmap#	(cb_bitmap) The bit pattern of a 16 x 16 pixels area representing the pattern. The 16 elements shall be separated by commas.

The following is an example of a white background pattern:

```
  background2:bmap#0xffff,0xffff,0xffff,0xffff,\
                   0xffff,0xffff,0xffff,0xffff,\
                   0xffff,0xffff,0xffff,0xffff,\
                   0xffff,0xffff,0xffff,0xffff
```

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

3.2 Action Items

The description of an action item consists of its name, which is the name of the item in most cases followed by a number. A colon separates the name from the description part, which consists of data items, action items, or in some cases some special actions.

3.2.1 Init

The init action consists of a list of actions to be performed upon initialization. They are executed in the specified order. The following things can be specified to be performed on initialization:

Item	Description
substitute	The keys used as substitute for the mouse. No keys will be set up if substitute is not present.
background	A new background pattern. The default pattern is used if no background is present.
pointer	The layout of the global mouse pointer. If no pointer is specified, the default mouse pointer is used.
terminal	Open a terminal window with a program running in it.

menu	The starting menu window. This must be specified.
inverse	Set the screen to inverse video. This is a special action and no number shall be given.
normal	Set the screen to normal video. This is a special action and no number shall be given.

Only one init action can be specified and therefore no init number shall be given.
An example:

```
  init:substitute1:menu3
```

3.2.2 Menu

The menu action describes a menu window, a pull down menu, etc.
The following things can be specified:

Item	Description
window	Data for the window to be used as menu. If the window already is open, wsh checks if the window already contains the desired strings and icons, and if so these are not set up once more. However if the contents is new, the new icons are set up and the new strings are displayed. One and only one window must be specified.
header	The header of the menu window. The header is optional.
choice	Describes the choices which it is possible to make from this menu. If no action is specified, at least one choice must be specified.
action	The specified action will be executed directly without waiting for a choice from the mouse. If any choices have been given, they are ignored.
string	Text and graphic contents of the menu window.
pstring	Text and graphic contents of the menu window.
lstring	Text and graphic contents of the menu window.
pointer	The layout of the mouse pointer when it points into this menu. If no pointer is specified, the global mouse pointer is used. The 'amsp' flag for the menu window must be set to make it possible to set up a private mouse pointer.

An example:

```
  menu5:window11:choice20:choice21:choice22:pstring13:lstring13
```

If both an action and choices are given, a warning message is issued.

3.2.3 Choice

The choice action connects an icon with the actions to be performed when that icon is chosen. The following two things must be specified in a choice:

Item	Description
icon	The icon.
action	The actions to be performed when the above icon is

chosen.

An example:

```
choice9:icon7:action17
```

3.2.4 Action

The action consists of a list of actions to be performed. The actions will be executed in the same order as they are specified. The following items may be specified in the list:

Item	Description
flags	New window flags for the current menu window.
substitute	Set new mouse substitute keys.
background	Set up a new background pattern.
pointer	Set up a new global mouse pointer.
terminal	Open a terminal window with a program running in it.
menu	Go to the specified menu.

The following special actions may be specified in an action list (no number shall be specified after these special actions):

Special action	Description
close	Close the current menu window.
restore	Restore the screen.
inverse	Set the screen to inverse video.
normal	Set the screen to normal video.
top	Move the current menu window to the top level.
turn	Turn the screen.
logout	Log out. This will only work if there are not any open terminal windows. Wsh takes care of checking this.

At least one 'menu' must be given. If several are given, a warning is issued. A warning also appears if a 'menu' does not end the list (in this case the actions after the 'menu' will never be executed).
An example:

```
action18:flags5:terminal10:close:menu7
```

3.2.5 Terminal

The terminal action describes a window to be used to run a program. The following can be specified in a terminal description:

window	Data for the window to be used as terminal. At most one may be specified. If no window is specified, the command will be executed with '/dev/null' as standard input, output, and error output.
header	Optional header for the terminal window.
zoomlist	Optional zoom list for the terminal window.
pointer	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

	possible to set up a private mouse pointer.
icon	Optional icons to be set up before the execution of the program starts.
string	Optional strings to be written to the terminal window before the execution of the program starts.
pstring	As above.
lstring	As above.
directory	An optional directory to move to before the execution of the program starts. If no directory is specified, the current directory for the program when it starts will be the same as the one where wsh were started from.
super	If present, the "super" channel will be open as file descriptor 3 in the program. This is a special action and no number shall be given.
wait	Causes wsh to wait for the command to finish. This is a special action and no number shall be given.
environ	Optional modification of the environment.
penviron	As above, but only for portrait mode.
lenviron	As above, but only for landscape mode.
command	Specifies the program to be executed in the terminal window.

An example:

```
  terminal15:window7:header7:pstring5:pstring6:lstring5:lstring6:\
            :directory11:environ3:environ4:penviron3:lenviron5:\
            :command23
```

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:

```
  window
  header
  icon
  pointer
  flags
  zoomlist
  substitute
  807          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

The syntax of wsh is:

```
wsh [-n] [<file>]
```

<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 are displayed in swedish instead.

3. Some Notes of the Behaviour of Wsh

- 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 should not be used both for menu and terminal icons.

- 'tsig' and 'nsig' are always set to 0 and the 'cbox' flag is cleared for menu windows. The reason is that wsh can not handle these things.

- The cursors are not moved to their home positions and the window is not cleared before the specified strings are displayed in a window. These things must, if necessary, be included in the strings. Be especially careful with strings which must be rewritten by wsh to update menu windows.

- Strings are always written in the specified order.

- All terminal windows are set up as controlling terminals, i.e. '/dev/tty' refers to the window.

- The processes running in different terminal windows belongs to different process groups.

- Only file descriptors 0, 1, and 2 (standard input, output, and error output) and sometimes 3 (the "super channel") are open when the command specified in a terminal description is executed.

- When the command in a terminal description is executed, all signals are set to default except those signals specified by 'tsig', 'nsig', 'rsig', and 'csig' which are ignored.

- The current directory for wsh is always the directory where it was started from. Terminals will initially have the same current directory if no 'directory' is specified.

- The command specified in terminals can be shell scripts and wsh automatically searches for the command in all the directories specified by the PATH environment variable.

- When handling the 'turn' special action, wsh checks that there are no windows open, except for menu windows. If not, all menu windows are closed and the window shell executes the 'init' action in the new screen mode.

- 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:

```
close(fd);
```

When the handler receives this request it sends hangup signals to all processes in the windows, resets the screen, and then executes 'whgo' again.
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).

4. Closing Windows

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

member of the union must be zero.

6.1   Create Window

To create a window the following request is used:

```
        Wincreat(fd, bp);
        int             fd;
        struct winstruc *bp;
```

'fd' is the file descriptor obtained from the open request and the structure winstruc looks like:

```
        typedef          short  pix_d;
        typedef          short  cur_d;
        typedef          char   sint;
        typedef unsigned short  word;
        typedef unsigned long   uflags;

        struct  winstruc
        {
                pix_d   wp_xorig;
                pix_d   wl_xorig;
                pix_d   wp_yorig;
                pix_d   wl_yorig;
                pix_d   wp_xsize;
                pix_d   wl_xsize;
                pix_d   wp_ysize;
                pix_d   wl_ysize;
                pix_d   wp_vxorig;
                pix_d   wl_vxorig;
                pix_d   wp_vyorig;
                pix_d   wl_vyorig;
                pix_d   wp_vxsize;
                pix_d   wl_vxsize;
                pix_d   wp_vysize;
                pix_d   wl_vysize;
                short   w_color;
                sint    w_border;
                char    wp_font;
                char    wl_font;
                char    w_curfont;
                sint    w_level;
                sint    w_uboxes;
                cur_d   w_xcur;
                cur_d   w_ycur;
                pix_d   w_xgcur;
                pix_d   w_ygcur;
                sint    w_tsig;
                sint    w_ntsig;
                sint    w_rsig;
                sint    w_csig;
                word    w_boxes;
                uflags  w_flags;
                sint    w_rstat;
                union
                {
                        long    w_xxx;
                } w_pad;
        };
```

The meaning of the structure members are:

wp_xorig	The x coordinate of the lower left corner of the virtual screen relative to the lower left corner of the screen. The coordinates are expressed in terms of pixels. If the lower left corner is to the left of the lower left corner of the screen, this value is negative. This coordinate is used in portrait mode.
wl_xorig	As 'wp_xorig', but used in landscape mode.
wp_yorig	The y coordinate of the lower left corner of the virtual screen in portrait mode.
wl_yorig	As 'wp_yorig', but used in landscape mode.
wp_xsize	The horizontal size of the virtual screen expressed in pixels in portrait mode.
wl_xsize	As 'wp_xsize', but used in landscape mode.
wp_ysize	The vertical size of the virtual screen expressed in pixels in portrait mode.
wl_ysize	As 'wp_ysize', but used in landscape mode.
wp_vxorig	The x coordinate of the lower left corner of the window (excluding the border) relative to the lower left corner of the virtual screen in portrait mode.
wl_vxorig	As 'wp_vxorig', but used in landscape mode.
wp_vyorig	The y coordinate of the lower left corner of the window in portrait mode.
wl_vyorig	As 'wp_vyorig', but used in landscape mode.
wp_vxsize	The horizontal size of the window in portrait mode.
wl_vxsize	As 'wp_vxsize', but used in landscape mode.
wp_vysize	The vertical size of the window in portrait mode.
wl_vysize	As 'wp_vysize', but used in landscape mode.
w_color	Background colour in the window (BLACK or WHITE).
w_border	The type of the border:
	NOBORDER - No border.
	SLBORDER - Single line border.
	DLBORDER - Double lines border.
	DSSSBORD - The left side is a double lines border and the rest of the sides are single line borders.
	SDSSBORD - The right side is a double lines border and the rest of the sides are single line borders.
	SSDSBORD - The upper side is a double lines border and the rest of the sides are single line borders.
	SSSDBORD - The lower side is a double lines border and

```
                                 the rest of the sides are single line
                                 borders.
                      DDSSBORD - The left and right sides are double lines
                                 borders and the upper and lower sides are
                                 single line borders.
                      DSDSBORD - The left and upper sides are double lines
                                 borders and the right and lower sides are
                                 single line borders.
                      DSSDBORD - The left and lower sides are double lines
                                 borders and the right and upper sides are
                                 single line borders.
                      SDDSBORD - The right and upper sides are double lines
                                 borders and the left and lower sides are
                                 single line borders.
                      SDSDBORD - The right and lower sides are double lines
                                 borders and the left and upper sides are
                                 single line borders.
                      SSDDBORD - The upper and lower sides are double lines
                                 borders and the left and right sides are
                                 single line borders.
                      DDDSBORD - The lower side is a single line border and
                                 the rest are double lines borders.
                      DDSDBORD - The upper side is a single line border and
                                 the rest are double lines borders.
                      DSDDBORD - The right side is a single line border and
                                 the rest are double lines borders.
                      SDDDBORD - The left side is a single line border and
                                 the rest are double lines borders.

        wp_font       The initial font in portrait mode. The font can be in
                      the range 'A' - 'Z'.

        wl_font       As 'wp_font', but used in landscape mode.

        w_curfont     The currently used font.

        w_level       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.

        w_uboxes      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.

        w_xcur        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.

        w_ycur        y coordinate for the text cursor position.

        w_xgcur       x coordinate for the graphic cursor. This one is
                      only used to return the initial position (which is
                      the lower left corner of the window).

        w_ygcur       y coordinate for the graphic cursor.

        w_tsig        The signal to be sent to the processes in the window
```

```
                              when it has moved to the top level (level zero). If
                              0, no signal will be sent.

          w_ntsig             As above, but signals are sent when the window moves
                              from the top level to a lower level.

          w_rsig              The signal to be sent to the processes in the window
                              when the window has changed in some way. If 0, no
                              signal will be sent.

          w_csig              The signal to be sent to the processes in the window
                              when the close box in the border is used. If 0, no
                              signal is sent, instead all requests to this window
                              will be terminated with bad status.

          w_boxes             Contains flags indicating which boxes shall be present
                              in the border:

                              BX_HSCR - Scroll left and right boxes and the
                                        horizontal visible indicator shall be
                                        present in the border.
                              BX_VSCR - Scroll up and down boxes and the vertical
                                        visible indicator shall be present in the
                                        border.
                              BX_CLOS - The close box shall be present in the
                                        border.
                              BX_SIZE - The size box shall be present in the border.
                              BX_MOVE - The move box shall be present in the border.
                              BX_ZOOM - The zoom box shall be present in the border.
                              BX_AVIS - The scroll boxes and the horizontal and
                                        vertical visible indicators are only visible
                                        if the whole virtual screen is not visible.
                              BX_BLOW - The "blow up" box shall be present in the
                                        border (see the Windflsz() request).
                              BX_HELP - The help box shall be present in the border
                                        (see the Winhelp() request).
                              BX_USER - Indicates that the value of the 'w_uboxes'
                                        member is significant.

          w_flags             Contains some flags:

                              PMODE      - Indicates that coordinates have been
                                           given for portrait mode.
                              LMODE      - Indicates that coordinates have been
                                           given for landscape mode.
                              SAVETEXT   - Save the text contents of the virtual
                                           screen.
                              SAVEBITMAP - Save the bitmap contents of the virtual
                                           screen (virtual bitmap) (reserved for
                                           future use).
                              OVERLAP    - The window is overlapped flag.
                              LOCK       - The window is locked on the highest level
                                           (level 0).
                              NOOVER     - The window must not be overlapped by
                                           another window.
                              NOCURSOR   - Cursor not visible.
                              NOMOVE     - The window must not be moved or change
                                           size.
                              ALLSCR     - The window must be the whole virtual
                                           screen.
                              SPECIAL    - A special window will be added on the
                                           lowest level. Special windows are always
```

	on lower levels than non-special windows and their level does not change when the level of other windows are changed. They can for example be used as menu windows.
KEYSCROLL	- Every time a key is pressed it is checked if the whole cursor is visible and if not the window is scrolled.
WRITSCROLL	- After each write request to the window, it is checked if the whole cursor is visible and if not the window is scrolled.
ALTMPNT	- Allocate space to store a mouse pointer which is used when we point to this window. Initially the mouse pointer will be the same as the global pointer. See the Winchmpnt() request.
RELATIVE	- The coordinates 'w_xorig' and 'w_yorig' are supposed to be relative to the lower left corner of the parent in this window group (see section 8).
NOCPIN	- Makes it impossible to copy text into this window using the text copy facility of the window handler.
NOCPOUT	- Makes it impossible to copy text from this window using the text copy facility of the window handler. Instead the status of the middle mouse button is reported on mouse position reports. Note that the middle button is only reported if this flag is set.
TXTSIZE	- The 'wp_xsize', 'wl_xsize', 'wp_ysize', 'wl_ysize', 'wp_vxorig', 'wl_vxorig', 'wp_vyorig', 'wl_vyorig', 'wp_vxsize', 'wl_vxsize', 'wp_vysize', and 'wl_vysize' members are supposed to be given in term of characters instead of pixels. Note that in this case 'wp_vxorig', 'wl_vxorig', 'wp_vyorig', and 'wl_vyorig' must be given relative to the upper left corner of the virtual screen.
WGROUP	- This window shall belong to a window group (see section 8).
REL_ULC	- This window shall follow its parent window relative the upper left corner of the parent (this flag has no effect if the window is not a child window).
REL_URC	- This window shall follow its parent window relative the upper right corner of the parent (this flag has no effect if the window is not a child window).
REL_LLC	- This window shall follow its parent window relative the lower left corner of the parent (this flag has no effect if the window is not a child window).
REL_LRC	- This window shall follow its parent window relative the lower right corner of the parent (this flag has no effect if the window is not a child window).

Note that at most one of the flags REL_ULC, REL_URC, REL_LLC, or REL_LRC may be set.
All these flags are single bits in the flags word.

```
                        Of these only the OVERLAP flag is non-significant when
                        creating a window.
                        All the remaining bits should be zero to guarantee
                        compatibility with future versions.

        w_rstat         Return status:
                        W_OK      - OK.
                        WE_ILPARA - an illegal parameter was specified.
                        WE_LORO   - the window can not be created because of
                                    another window with the NOOVER or LOCK
                                    flag set.
                        WE_ALRCR  - the window has already been created.
                        WE_ALLSCR - the whole virtual screen is not visible
                                    and the ALLSCR flag is set.
                        WE_NOMEM  - enough memory does not remain to create
                                    the window.
                        WE_FATHER - the window has the RELATIVE flag set, but
                                    there is no parent window.
                        WE_ILMOD  - the coordinates for the current screen
                                    mode has not been given, e.g. the screen
                                    is in landscape mode and the LMODE flag
                                    is not set.
                        WE_NOFONT - the specified default font can not be
                                    loaded.
```

Of the above members, only the following are used when a window is created:

wp_xorig or wl_xorig, wp_yorig or wl_yorig, wp_xsize or wl_xsize, wp_ysize or wl_ysize, wp_vxorig or wl_vxorig, wp_vyorig or wl_vyorig, wp_vxsize or wl_vxsize, wp_vysize or wl_vysize, w_color, w_border, wp_font or wl_font, w_tsig, w_ntsig, w_rsig, w_csig, w_boxes, w_flags

On exit the values of these members remains the same, except for some adjustments that may occur in order to make the window fit, etc. The other members have on exit received their initial values.

6.2 Move Window to Level Zero

The level zero window is the window that receives the keyboard input. The request

```
Winlevel(fd, bp)
int             fd;
struct winlevel *bp;
```

is used to move a window which does not belong to a window group to the zero level. If the window indicated by 'fd' belongs to a window group, the whole group is moved to the top without altering the relative levels inside the group.
The winlevel structure looks like:

```
typedef char    sint;

struct  winlevel
{
        sint    l_rstat;
        union
        {
```

```
                        long    l_xxx;
                } l_pad;
        };
```

where 'l_rstat' is the return status:

```
        W_OK       - everything is well.
        WE_NOTCR   - the window has not been created yet.
        WE_SPECIAL - the window can not be moved to the top because
                     it is a special window.
        WE_LORO    - the level can not be changed because of another
                     window with the LOCK or NOOVER flags set.
```

6.3   Move Window to the Top Level of its Window Group

To move a window, belonging to a window group, to the top level of the group, use the request:

```
        Winllev(fd, bp)
        int             fd;
        struct  winlevel *bp;
```

'fd' is the file descriptor for the window and the winlevel structure was described in section 6.2.

6.4   Alter a Window

To alter the size, position, etc. of a window, the request

```
        Winalter(fd, bp);
        int             fd;
        struct winstruc *bp;
```

is used. If the window is a parent of a window group, all the children are also moved according to the flags REL_ULC, REL_URC, REL_LLC, and REL_LRC. If none of these flags are set for a child window, the child is not moved.
The winstruc structure was described in section 6.1. On entry to this request, the following structure member values are significant:

```
        wp_xorig or wl_xorig, wp_yorig or wl_yorig, wp_vxorig or
        wl_vxorig, wp_vyorig or wl_vyorig, wp_vxsize or wl_vxsize,
        wp_vysize or wl_vysize
```

Further the PMODE and LMODE flags in 'w_flags' are used to check that the data is relevant and if the TXTSIZE flag is set, the coordinates and sizes are interpreted in units of characters. The size of the current default font is used.
The remaining parameters can not be changed using this request, but the current values of them are returned.
'w_rstat' is the return status:

```
        W_OK      - all is well.
        WE_NOTCR  - the window has not been created yet.
        WE_ILPARA - an illegal parameter value was used.
        WE_LORO   - the window can not be altered because of another
                    window with the LOCK or NOOVER flags set.
        WE_ALLSCR - the whole virtual screen will not be visible and
```

```
                  the ALLSCR flag for the window is set.
     WE_NOMOVE - it is not allowed to change the location or the
                  size of the window (the NOMOVE flag is set).
     WE_ILMOD  - data for the current screen mode is not present.
```

6.5 Alter a Window without Affecting Child Windows

This request is identical to the Winalter() request, except that if the specified window is a parent of a window group, its child windows are not moved.
The request is:

```
Win1alter(fd, bp)
int             fd;
struct  winstruc *bp;
```

6.6 Set up Default Size and Location for a Window

When the "blow up" box is used the size and location of the window toggles between the default size and location and the size and location it had before it was altered to the default.
When a window is created, its initial default size and location will be the same as the initial size and location of the window.
When the default font is changed, the default size and location will be set to the same as the size and location of the window after the default font has been changed.
To set up another default size and location, use the request:

```
Windflsz(fd, bp)
int             fd;
struct  winstruc *bp;
```

The winstruc structure was described in section 6.1. On entry to this request the following structure members are significant:

```
wp_xorig or wl_xorig, wp_yorig or wl_yorig, wp_vxorig or
wl_vxorig, wp_vyorig or wl_vyorig, wp_vxsize or wl_vxsize,
wp_vysize or wl_vysize
```

Further the PMODE and LMODE flags in 'w_flags' are used to check that the data is relevant and if the TXTSIZE flag is set the coordinates and sizes are interpreted in units of characters. The size of the current default font is used.
The return status - 'w_rstat' - is:

```
W_OK       - all is well.
WE_NOTCR   - the window has not been created yet.
WE_ILMOD   - data for the current screen mode is missing.
WE_ILPARA  - an illegal value was specified.
```

6.7 Alter Window Flags

To alter the flags in the 'w_flags' word for a window, use the request:

```
Winflags(fd, bp);
```

```
        int             fd;
        struct  flgstruc *bp;
```

The flgstruc structure looks like:

```
        typedef unsigned long   uflags;
        typedef          char   sint;

        struct  flgstruc
        {
                uflags  f_flags;
                sint    f_rstat;
                union
                {
                        long    f_xxx;
                } f_pad;
        };
```

'f_flags' is the new flags for the window.
The following flags may be altered: LOCK, NOOVER, NOCURSOR, NOMOVE, ALLSCR, KEYSCROLL, WRITSCROLL, NOCPIN, NOCPOUT, REL_ULC, REL_URC, REL_LLC, and REL_LRC.
The following flags are ignored: PMODE, LMODE, SAVETEXT, SAVEBITMAP, OVERLAP, SPECIAL, ALTMPNT, RELATIVE, TXTSIZE, and WGROUP.
The the bits not used in the flags word should be zero to guarantee compatibility with future versions.
'f_rstat' is the return status:

```
        W_OK      - everything is OK.
        WE_LORO   - the flags can not be altered in this way because
                    the window is overlapped or it is not on the top
                    level.
        WE_ALLSCR - the whole virtual screen is not visible and the
                    ALLSCR flag was set.
```

6.8 Get Window Status

To get the current status of a window, use the request

```
        Winstat(fd, bp);
        int             fd;
        struct winstruc *bp;
```

The winstruc structure was described in section 6.1.
On exit all the members are set to their current values. Only one of portrait or landscape mode coordinates and font is returned, depending on the mode of the screen. Which one is indicated by the PMODE and LMODE flags.
The return status 'w_rstat' is:

```
        W_OK     - all is well.
        WE_NOTCR - the window has not been created yet.
```

6.9 Insert a Header in a Window Border

To insert a header, such as the program name, in the border of a window, use the request

```
        Winheader(fd, bp);
        int             fd;
        struct headstruc *bp;
```

where the headstruc structure looks like:

```
        typedef unsigned short  word;

        struct  headstruc
        {
                char    h_hdr[HDRSIZE];
                word    h_flags;
                union
                {
                        long    h_xxx;
                } h_pad;
        };
```

'h_hdr[]' is the header string, 'h_flags' contains some flags:

```
        H_INVHD  - Invert the window header (relative the window
                   background).
        H_INVTOP - Invert the top window header (relative H_INVHD).
```

The remaining bits should be zero to guarantee compatibility with future versions.
Note that the header can be added before the window is created.

6.10   Icon Support

The window handler can automatically take care of decoding commands given by first pointing to an icon, menu item, etc. and then pressing an appropriate key on the mouse or the keyboard.
The request

```
        Winicon(fd, bp);
        int             fd;
        struct  winicon *bp;
```

is used to specify that when the pointer points inside a specified area in the window, a specified code sequence shall be sent to the calling process by putting it in the keyboard input buffer for the window.
The winicon structure looks like:

```
        typedef         short  pix_d;
        typedef unsigned short  word;
        typedef         char   sint;

        struct  winicon
        {
                pix_d   ip_xorig;
                pix_d   il_xorig;
                pix_d   ip_yorig;
                pix_d   il_yorig;
                pix_d   ip_xsize;
                pix_d   il_xsize;
                pix_d   ip_ysize;
                pix_d   il_ysize;
                char    i_cmdseq[ICONSEQLEN];
                word    i_flags;
                sint    i_rstat;
                union
                {
                        long    i_xxx;
                } i_pad;
        };
```

'ip_xorig', 'il_xorig' and 'ip_yorig', 'il_yorig' is the lower left 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 area in portrait and landscape mode, respectively.
'i_cmdseq[]' is the sequence to be sent to the calling process (it can be of zero length).
'i_flags' contains some flags indicating the type of icon and some attributes:

I_PMODE	- Portrait mode coordinates are given.
I_LMODE	- Landscape mode coordinates are given.
I_PRESS	- Send the sequence when the mouse pointer points to the area and the left button is pressed.
I_RELEASE	- Send the sequence when the mouse pointer points to the area and the left button is released.
I_INVERT	- Invert the area occupied by the icon when the mouse pointer is pointing to it.
I_ENTER	- The sequence is sent when the mouse pointer moves into the area. The area does not have to be visible. The I_INVERT flag is ignored.
I_LEAVE	- As I_ENTER but the sequence is sent when we leave the area.
I_REMOVE	- The icon is removed when the sequence has been sent.
I_RQST	- The sequence is sent only if there is a pending read request to the window.
I_SETCHK	- When I_ENTER and/or I_LEAVE is set, it is checked if the mouse pointer is inside or outside, respectively, the specified area, and if so the sequence is sent immediately.
I_LZERO	- The sequence is sent only if it is the level zero window.
I_TEXT	- The coordinates and sizes of the icon is supposed to be given in term of characters instead of pixels. Note that 'ip_xorig' and 'ip_yorig' or 'il_xorig' and 'il_yorig' in this case are interpreted as the character position relative the upper left corner of the virtual screen. When the default font is changed, the locations and sizes of icons set up with this flag set are adjusted.

The remaining bits should be zero to guarantee compatibility with future versions.
Note that if no one of I_PRESS, I_RELEASE, I_ENTER, or I_LEAVE is given, I_PRESS is assumed. I_ENTER and I_LEAVE overrides I_PRESS and I_RELEASE.

The return status 'i_rstat' is:

W_OK	- everything is well.
WE_NOTCR	- the window is not created yet.
WE_ILPARA	- any of the input parameters are illegal.

```
        WE_NOICON - no memory left for the new icon.
        WE_ONICON - the icon will come above another icon in the same
                    window.
        WE_ILMOD  - no coordinates are given for the current screen
                    mode.
```

6.11 Remove Icon's

To remove all set up icon's for a window, use the request:

```
        Rmicons(fd);
        int     fd;
```

6.12 Mouse Substitute Keys

To make it possible to use the window handler without a mouse, the different functions supported by the mouse can be simulated by function and other special keys on the ABC99 keyboard (these keys generates codes with the most significant bit set).
To specify these keys, use the request:

```
        Winmsub(fd, bp);
        int             fd;
        struct  substit *bp;
```

The file descriptor used must be the one obtained when the window handler was activated (the first open request to the handler). The structure substit looks like:

```
        typedef char    sint;

        struct  substit
        {
                sint            c_initflg;
                unsigned char   c_keys[SUBSTKEYS];
                unsigned char   c_step;
                unsigned char   c_lstep;
                union
                {
                        long    c_xxx;
                } c_pad;
        };
```

The meaning of the different members are:

c_initflg	If ON the mouse simulation keys will be enabled after this request. If OFF they will initially be disabled.
c_keys[]	The keys used as substitue for the mouse.
c_step	Step for normal mouse pointer move (no. of pixels).
c_lstep	Step for long mouse pointer move (no. of pixels).

The index for the different keys in the 'c_keys[]' array are:

S_ONOFF	The key used to toggle the mouse simulation keys on or off. When off, the keys behaves as normal (except 'S_ONOFF').
S_MPU	Move mouse pointer up.
S_MPD	Move mouse pointer down.
S_MPL	Move mouse pointer left.

S_MPR	Move mouse pointer right.
S_MPUL	Move mouse pointer up - left.
S_MPUR	Move mouse pointer up - right.
S_MPDL	Move mouse pointer down - left.
S_MPDR	Move mouse pointer down - right.
S_LMPU	Move mouse pointer up long.
S_LMPD	Move mouse pointer down long.
S_LMPL	Move mouse pointer left long.
S_LMPR	Move mouse pointer right long.
S_LMPUL	Move mouse pointer up - left long.
S_LMPUR	Move mouse pointer up - right long.
S_LMPDL	Move mouse pointer down - left long.
S_LMPDR	Move mouse pointer down - right long.
S_PCMD	Point to command key (replaces the left key on the mouse).
S_CHWIN	Change window level key (replaces the right key on the mouse).
S_MCA	Mark text area to copy (replaces the middle key on the mouse).

Pressing and releasing a button on the mouse is replaced by pressing the chosen keyboard key twice.
Note that no keys will be occupied by these keys if this request has not been issued.

6.13 Alter the Background Pattern

To alter the pattern of the background, use the request:

```
Winchbg(fd, bp)
int             fd;
struct  chbgstruc *bp;
```

'fd' must be the file descriptor obtained when the window handler was activated.
The chbgstruc structure looks like:

```
typedef unsigned short  word;

struct  chbgstruc
{
        word    cb_bitmap[BGPSIZE];
        union
        {
                long    cb_xxx;
        } cb_pad;
};
```

'cb_bitmap[]' is the bit pattern of a 16 x BGPSIZE pixels area which will be repeated all over the background.
Note that the most significant bit in a "word" is displayed to the left on the screen.

6.14 Get the Visible Parts of a Window or the Background

To get the visible parts of a window or the background, use the request:

```
        Wingetvis(fd, bp, bc)
        int             fd;
        struct  buffer  *bp;
        int             bc;
```

'fd' is the file descriptor for the window, or the file descriptor obtained when the window handler was activated if the visible parts of the background are desired.
'bc' is the size of the buffer structure.
The buffer structure looks like:

```
        struct  buffer
        {
                struct  visdes  v;
                struct  rectdes b[VSIZE];
        };
```

The visdes structure is a parameter structure and looks like:

```
        typedef char    sint;

        struct  visdes
        {
                short   v_nrect;
                sint    v_rstat;
                union
                {
                        long    v_xxx;
                } v_pad;
        };
```

The rectdes structure describes one rectangle which the visible part of the virtual screen or the background can be divided into:

```
        typedef short   pix_d;

        struct  rectdes
        {
                pix_d   r_xorig;
                pix_d   r_yorig;
                pix_d   r_xsize;
                pix_d   r_ysize;
        };
```

where 'r_xorig' and 'r_yorig' are the x and y coordinates respectively of the lower left corner of the rectangle. 'r_xsize' and 'r_ysize' are the width and height, respectively, of the rectangle.
When this request is executed the 'v_nrect' member of visdes should contain the number of rectdes structures (VSIZE) in the buffer structure. The request returns the actual number of rectangles that the virtual screen (or the background) can be divided into in 'v_nrect'.
'v_rstat' is the return status:

```
        W_OK     - Ok
        WE_NOTCR - The window has not been created yet.
        WE_SPACE - Not enough space to hold the rectangles (i.e. VSIZE
                   is too small).
```

6.15   Inverse Video
        =============

The request:

```
        Winivideo(fd)
        int     fd;
```

changes the screen to inverse video. 'fd' must be the file descriptor obtained when the window handler was activated.

6.16 Normal Video

The request:

```
        Winnvideo(fd)
        int     fd;
```

restores the screen to normal video. 'fd' must be the file descriptor obtained when the window handler was activated.

6.17 Make the Cursor Visible in the Window

To make the cursor visible in the window, use the request:

```
        Wincurvis(fd)
        int     fd;
```

If the whole cursor is not visible, the window is scrolled.

6.18 Change Mouse Pointer

To change the layout of the mouse pointer, use the request:

```
        Winchmpnt(fd, bp)
        int             fd;
        struct  npstruc *bp;
```

If 'fd' is the file descriptor obtained when the window handler was activated, the global mouse pointer is altered. Otherwise the mouse pointer for the window indicated by the file descriptor is altered (in this case, the ALTMPNT flag for the window must be set).
The npstruc structure looks like:

```
        typedef         short  pix_d;
        typedef unsigned long   dword;
        typedef unsigned char   byte;
        typedef          char   sint;

        struct  npstruc
        {
                pix_d   np_xsize;
                pix_d   np_ysize;
                pix_d   np_xpnt;
                pix_d   np_ypnt;
                dword   np_and[MPSIZE];
                dword   np_or[MPSIZE];
                byte    np_flags;
                sint    np_rstat;
                union
                {
                        long    np_xxx;
                } np_pad;
        };
```

'np_xsize' and 'np_ysize' are the width and height, respectively, of the new mouse pointer. The maximal width is 32 pixels and the height MPSIZE pixels.
'np_xpnt' and 'np_ypnt' are the pixel which is the pointing part of the mouse pointer. It shall be specified relative the upper left corner of the mouse pointer.
'np_and[]' and 'np_or[]' are masks used to construct the mouse pointer.
Each pixel row of the mouse pointer is constructed by the operation:

```
        (x & np_and[prow]) | np_or[prow]
```

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.
'np_flags' is reserved for future use and should be zero to guarantee compatibility with future versions.
'np_rstat' is the return status:

```
        W_OK      - Ok.
        WE_ILPARA - An illegal value was specified.
        WE_NOTCR  - The window has not been created yet.
        WE_NOMP   - The ALTMPNT flag for the window is not set, and
                    therefore the mouse pointer can not be changed.
```

6.19   Get Number of Open Windows

To find out how many windows which are open and/or created, use the request:

```
        Wincnt(fd, bp)
        int             fd;
        struct  nwstruc *bp;
```

'fd' is the file descriptor obtained when the window handler was activated or the file descriptor for a window.
The nwstruc structure looks like:

```
        struct  nwstruc
        {
                short   nw_open;
                short   nw_created;
                union
                {
                        long    nw_xxx;
                } nw_pad;
        };
```

'nw_open' is the number of windows currently open and 'nw_created' is the number of windows currently created (and opened).

6.20   Restore Screen

To restore the screen, i.e. rewrite the whole screen, use the request:

```
Winrestor(fd)
int     fd;
```

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

6.21 Get Text Contents of Window

To get the text contents of a window, use the request:

```
Wingettxt(fd, bp, bc)
int             fd;
struct  buffer  *bp;
int             bc;
```

'fd' is the file descriptor for the window. The structure buffer consists of a parameter structure followed by a buffer with space to hold the desired text contents:

```
struct  buffer
{
        struct  txtstruc s;
        char            b[BSIZE];
};
```

The txtstruc structure looks like:

```
typedef short   cur_d;
typedef char    sint;

struct  txtstruc
{
        cur_d   tx_row;
        cur_d   tx_col;
        cur_d   tx_rcnt;
        cur_d   tx_ccnt;
        sint    tx_rstat;
        union
        {
                long    tx_xxx;
        } tx_pad;
};
```

'tx_row' is the row number of the first row to be read and 'tx_col' the number of the first column. 'tx_rcnt' and 'tx_ccnt' is the number of rows and columns, respectively, to be read. BSIZE must be at least tx_rcnt * tx_ccnt.
'tx_rstat' is the return status:

```
W_OK      - Everything is ok.
WE_TSAVE  - The text contents of the window is not saved.
WE_ILPARA - Illegal parameters was given.
```

6.22 Test if Window Handler is Activated

To test if the window handler is activated, use the request:

```
        Wintest(fd)
        int     fd;
```

'fd' is the file descriptor for a window or the one obtained when the handler was activated.
If a negative value is returned, the window handler is not present.

6.23 Set Initial Driver and Terminal Parameters

This request is used to set the initial driver and terminal parameters for windows. The request is:

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

'fd' must be the file descriptor obtained when the window handler was activated.
The wininit structure looks like:

```
        typedef unsigned long   t_stop;
        typedef unsigned short  word;

        struct  wininit
        {
                t_stop  td_tbstop[TSTOPSIZE];
                word    td_term;
                struct
                {
                        unsigned short  c_iflag;
                        unsigned short  c_oflag;
                        unsigned short  c_cflag;
                        unsigned short  c_lflag;
                        char            c_line;
                        unsigned char   c_ccs[8];
                } td_driver;
                union
                {
                        long    td_xxx;
                } td_pad;
        };
```

'td_tbstop[]' contains the tab stops. A set bit indicates a tab stop.
The least significant bit of the first element corresponds to the first character position of a row.
'td_term' contains initial VT-100 terminal flags:

TD_NL	linefeed newline mode.
TD_WRAP	auto wrap mode.
TD_ORIGIN	origin mode.
TD_USCORE	underscore character attribute.
TD_REVERSE	reverse character attribute.
TD_SCREEN	screen mode.
TD_CUNDER	underline cursor.
TD_NONBLNK	non-blinking cursor.
TD_PHASE	phased pattern mode.
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:

```
        Winzoom(fd, bp)
        int             fd;
        struct  zoomlst *bp;
```

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

```
        typedef unsigned char   byte;
        typedef          char   sint;

        struct  zoomlst
        {
                char    zp_list[ZOOMSIZE];
                char    zl_list[ZOOMSIZE];
                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.

'z_flags' contains some flags:

```
        Z_PMODE - Portrait mode zoom list is given.
        Z_LMODE - Landscape mode zoom list is given.
```

The remaining bits should be zero to guarantee compatibility with future versions.
The return status 'z_rstat' is:

```
        W_OK      - everything is ok.
        WE_ILPARA - an illegal font was specified.
        WE_ILMOD  - no list is given for the current screen mode.
```

Note that this request can be used before the window has been created.

6.26   Change the Default Font for a Window

To change the default font for a window, use the request:

```
        Winndchr(fd, bp);
        int             fd;
        struct  dfltchr *bp;
```

'fd' is the file descriptor for the window and the dfltchr structure looks like:

```
        typedef         short  cur_d;
        typedef unsigned char   byte;

        struct  dfltchr
        {
                char    dcp_font;
                char    dcl_font;
                cur_d   dcp_x;
                cur_d   dcl_x;
                cur_d   dcp_y;
                cur_d   dcl_y;
                byte    dc_rstat;
                union
                {
                        long    dc_xxx;
                } dc_pad;
        };
```

'dcp_font' and 'dcl_font' are the new default font in portrait and landscape mode, respectively. If the specified font is zero, the next font in the zoom list is used.
'dcp_x', 'dcp_y', 'dcl_x', and 'dcl_y' is the character coordinates in portrait and landscape mode, respectively, for the middle character in the window after the default font has been changed.
'dc_flags' contains some flags:

```
        Z_PMODE - Data has been given for portrait mode.
        Z_LMODE - Data has been given for landscape mode.
```

The remaining bits should be zero to guarantee compatibility with future versions.
'dc_rstat' is the return status:

```
        W_OK      - everything is ok.
        WE_NOTCR  - the window has not been created yet.
        WE_ILMOD  - no data is given for the current screen mode.
        WE_ILPARA - an illegal font and/or illegal character
                    coordinates were given.
        WE_TSAVE  - the text contents of the virtual screen is not
                    saved.
        WE_ALLSCR - the ALLSCR flag for the window is set.
        WE_NOMOVE - the NOMOVE flag for the window is set.
        WE_NOFONT - the specified font does not exist.
```

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 same number of character rows and columns.

6.27 Turn the Screen

To turn the screen from portrait to landscape mode or vice versa, use the request:

```
        Winturn(fd, bp)
        int             fd;
        struct  modstruc *bp;
```

All channels, except the one obtained when the window handler was activated, must be closed.
'fd' must be the file descriptor obtained when the window handler was activated. The modstruc structure looks like:

```
        typedef char    sint;

        struct  modstruc
        {
                sint    m_mode;
                sint    m_rstat;
                union
                {
                        long    m_xxx;
                } m_pad;
        };
```

'm_mode' will on return be M_PORT if the new mode is portrait mode or M_LAND if it is landscape.
'm_rstat' is the return status:

```
        W_OK    - everything is ok.
        WE_OPEN - there are windows open.
```

6.28 Get Screen Mode

To get the current screen mode (portrait or landscape), use the request:

```
        Winmode(fd, bp)
        int             fd;
        struct  modstruc *bp;
```

'fd' is the file descriptor obtained when the window handler was activated or the file descriptor for a window. The modstruc structure was described in section 6.27. The 'm_mode' member contains the current mode (M_PORT or M_LAND) and 'm_rstat' is always W_OK.

6.29 Add a User Defined Box

User defined boxes are 16x16 pixels boxes in the left side of the window border. When the mouse pointer points to a user box and the left mouse button is pressed, a signal is sent to the process(es) running in the window.
When a window is created, the maximal number of user defined boxes for the window must be specified (see the Wincreat() request).
To set up a user defined box, use the request:

```
Winubox(fd, bp)
int             fd;
struct  userbox *bp;
```

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

```
typedef unsigned short  word;
typedef unsigned char   byte;
typedef          char   sint;

struct  userbox
{
        word    bx_bmap[UBOXSIZE];
        short   bx_sig;
        byte    bx_flags;
        sint    bx_rstat;
        union
        {
                long    bx_xxx;
        } bx_pad;
};
```

'bx_bmap[]' contains the bitmap for the box. Note that the most significant bit in a "word" is displayed to the left on the screen.
'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 compatibility with future versions.
'bx_rstat' is the return status:

```
W_OK      - all is well.
WE_NOTCR  - the window has not been created yet.
WE_SPACE  - the maximal number of user defined boxes have
            already been set up.
WE_ILPARA - an illegal signal number was specified.
```

6.30 Alter Help Box Sequence

The help box is a box in the upper side of the border containing a question mark which when used puts a character sequence on the key-board input buffer. The intention is that all programs use this facility so help can be requested in a similar manner in all programs.
When a window is opened, the help box sequence is initialized to a single question mark (?). To alter this to another sequence, use the request:

```
        Winhelp(fd, bp)
        int             fd;
        struct  helpst  *bp;
```

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

```
        typedef unsigned short  word;

        struct  helpst
        {
                char    hlp_seq[HLPSIZE];
                word    hlp_flags;
                union
                {
                        long    hlp_xxx;
                } hlp_pad;
        };
```

'hlp_seq[]' is the new help box sequence. 'hlp_flags' is reserved for future use and should be zero to guarantee compatibility with future versions of the window handler.
Note that the help box sequence can be altered before the window has been created.

6.31 Keyboard Input Signal

To make it possible to know when there is something to read from the 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 reading the keyboard buffer will not lead to wait.
The request is:

```
        Winkysig(fd, bp)
        int             fd;
        struct  kysigst *bp;
```

'fd' is the file descriptor for the window and the kysigst structure looks like:

```
        struct  kysigst
        {
                sint    ks_sig;
                byte    ks_flags;
                sint    ks_rstat;
                union
                {
                        long    ks_xxx;
                } ks_pad;
        };
```

'ks_sig' is the signal to be sent. If zero, no signals are sent.
'ks_flags' is reserved for future use and should be zero to guarantee compatibility with future version.
'ks_rstat' is the return status:

```
        W_OK      - everything is well.
        WE_ILPARA - an illegal signal was specifiead.
```

6.32   Read the Contents of the Picture Memory

To read the contents of the picture memory for a window or the whole screen, use the request:

```
Wpictrd(fd, bp, bc);
int             fd;
struct  buffer  *bp;
int             bc;
```

'fd' is the file descriptor for the window or, if the contents of the whole screen is desired, the file descriptor obtained when the window handler was activated. The buffer structure consists of a parameter structure followed by a buffer big enough to hold the contents of the specified picture memory area:

```
typedef unsigned char   byte;

struct  buffer
{
        struct  wpictblk p;
        byte            b[BSIZE];
};
```

The wpictblk structure looks like:

```
typedef short   pix_d;

struct  wpictblk
{
        pix_d   p_xaddr;
        pix_d   p_yaddr;
        pix_d   p_width;
        pix_d   p_height;
        union
        {
                long    p_xxx;
        } p_pad;
};
```

'p_xaddr' and 'p_yaddr' are the x and y pixel coordinates, respectively, of the lower left corner of the area to read. 'p_width' is the pixel width of the area and 'p_height' the pixel height.
BSIZE must be at least p_height * (p_width + 7) / 8.
Data areas in buffer.b[] corresponding to non-visible areas of a virtual screen will contain zeroes, i.e. cleared bits.
Note that the most significant bit in a byte is displayed to the left on the screen.
WARNING: At the moment this request is extremely slow and the computer seems to hang up while this request is served.

6.33   Alter the Spray Mask

This request changes the 32 times 32 pixels pattern used by the spray escape sequence (see wh_escapes.doc).
The request is:

```
Spraymask(fd, bp)
```

```
        int             fd;
        struct  sprayst *bp;
```

'fd' is the file descriptor for the window and the sprayst structure looks like:

```
        typedef unsigned long   dword;

        struct  sprayst
        {
                dword   sp_mask[8*sizeof(dword)];
        };
```

where 'sp_mask[]' contains the bit pattern for the spray mask. Note that the most significant bit in a "dword" is displayed to the left on the screen.

7. Other I/O Control Commands

This is a list of I/O control requests which are identical or similar to their counterparts in the tty device driver:

PFNKLD	Load ABC99 function keys. The file descriptor can be both the one for a window and the one obtained when the window handler was activated.
PFNKRD	Read ABC99 function keys. The file descriptor can be both the one for a window and the one obtained when the window handler was activated.
PTOKBD	Write data to the ABC99 keyboard. The file descriptor must be the one obtained when the window handler was activated.
TIOCGETP	Fetch the basic parameters for the terminal (v7).
TIOCSETP	Flush and then set the basic parameters (v7).
TIOCSETN	Set the basic parameters (no flush) (v7).
TIOCEXCL	Set "exclusive-use" mode (v7).
TIOCNXCL	Turn off "exclusive-use" mode (v7).
TIOCFLUSH	Flush input and output queues (v7).
TIOCSETC	Set the special characters (v7).
TIOCGETC	Get the special characters (v7).
FIORDCHK	Check if any character on input (v7).
TCSETAF	Wait for output to drain, then flush the input queues, and set the parameters for the terminal.
TCSETAW	As above, but do not flush the input queues.
TCSETA	Set the parameters for the terminal.
TCGETA	Get the parameters for the terminal.
TCFLSH	Flush the input, output, or both the input and output queues.

It should be noted that the set up of the ABC99 function keys is common for all windows. Hence the PFNKLD and PFNKRD requests should be used carefully.

8. Window Groups

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

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 text contents and regards text as graphics:

i) 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:

i) Send the Reset to Initial State escape sequence (ESC c) to the window or
ii) 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.