USERS MANUAL
ABC1600 WINDOW HANDLER

PREFACE

The ABC1600 WINDOW HANDLER is a powerful help to fully utilize the graphic capacity of the microprocessor system ABC1600. It is specially useful when making menu windows. An example of such a menu if the one which is displayed when the Window SHell (WSH) is started.

We hope that this users manual shall help you to get familiar with the graphics and design your own menu windows as quickly as possible.

Motala in November 1985

LUXOR DATORER AB

CONTENTS

SUB-CONTENTS

1 INTRODUCTION

1.1 HOW TO USE THIS MANUAL

This manual consists of four major parts. They are presented briefly in the following chapter (1.2). However, the most important chapters when using the manual are the chapters two and three, describing the basic operations and the requests necessary when programming.

For quickest possible wiew over the function of the window handler, study the examples in chapter five. But, very important it is of course to begin with the presentation (1.2).

In the end of this manual there is a set of three opens a window in both 'por is more complex and involves icons. Example three opens a window with status of a specified terminal - i e emulates the specified terminal.

For optimal survey, the CONTENTS plan for this manual is limited to one page. Then, if a special command is looked for, there is a complete listing of the command headings of the chapters three and six under the heading SUB-CONTENTS. If you already know the name of the command or keyword, use the INDEX instead.

The USERS MANUAL can also be used as a reference document because the INDEX includes the most important expressions as well as keywords and request statements.

1.2 PRESENTATION

As indicated by the name, the ABC1600 WINDOW HANDLER is implemented as a window handler under ABCenix. Its calls are specially designed for creating and manipulating windows with minimum effort. Typical examples are moving windows and returning the status of a window.

The main features of the ABC1600 WINDOW HANDLER are:

- Mouse control. Using the mouse as input source, most user actions are simplified and made faster.

- Command convertions. Using the mouse pointing on a specified area inside a window (an icon), the ABC1600 WINDOW HANDLER converts it to a complete command sequence.

- Moving text between windows.

The first of the four major chapters is number two - a detailed description of basic operations on windows, such as starting them and writing in them. Chapter three describes the function of the window requests - the commands that are used as calls from programs and controls the general window operations. In addition to these commands there are others which for example affects the I/O and the virtual screen. They are described in chapter four. Chapter six regards the window utility commands which, unlike the window requests, are used directly from the keyboard.

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 zero and it receives all the input from the keyboard. All the other windows are at lower levels. This means that the window one step from the top is at level one 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 zero. When this is done, all windows previously at higher levels than the new level zero window automatically are moved one level down. The level zero window can also be moved to the bottom, making all other windows automatically move one level up. The output from the processes connected to a certain window are always sent to that window, regardless of its level.

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 chapter two of the swedish documentation 'ANVÄNDARHANDLEDNING ABC1600 FÖNSTERHANTERARE' for further details regarding the escape sequences.

2 BASIC OPERATIONS

2.1 STARTING WINDOWS

The window handler is started by 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. The request for opening (activating) the window handler runs

```
        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.

When starting 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.

2.2 TERMINATING WINDOWS

The request for closing (disactivating) the window handler runs

```
        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'.

2.3 OPENING WINDOWS

When the window handler has been activated, windows can be opened by issuing an opening request to the handler which runs

```
        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 and is used for writing to, reading from, sending I/O control requests to, and for window closing

To acctually create the window on the screen, the Wincreat request is used (see 3.1, page 9)

2.4 CLOSING WINDOWS

To close a window, a closing trequest shall be sent to the handler with the file descriptor obtained when the window was opened. The request runs

```
        close(fd);
```

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

2.5 WRITING IN WINDOWS

When writing in a window the standard write system call can be used with the same file descriptor obtained as when the window was opened. The request runs

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

2.6 READING FROM WINDOWS

To read from a window (get input from the keyboard), the read system call can be used. The request runs

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

3 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 statements 'union' included in most of the structures below are reserved for future use. To guarantee compatibility with future versions, the statement 'union' must be zero in all structures where 'union' is included.

In all the following window requests, 'fd' is the file descriptor obtained from the 'open' request. This specific 'fd' must be used when included in a window request as a parameter.

3.1 CREATE WINDOW

The request for creating a window runs

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

'fd' is the file descriptor obtained from the open request.

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
ä
        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;
                    å w_pad;
                å;
```

All these flags are single bits in the flags unit descriptor 'word'. Of these only the 'OVERLAP' flag is non-significant when creating a window.

KEYWORD - DESCRIPTION

wp_xorig	'Portrait mode': The x coordinate of the lower left corner of 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.
wl_xorig	As 'wp_xorig', but used in 'landscape mode'.
wp_yorig	'Portrait mode': The y coordinate of the lower left corner of the virtual screen.
wl_yorig	As 'wp_yorig', but used in 'landscape mode'.
wp_xsize	'Portrait mode': The horizontal size of the virtual screen expressed in pixels.
wl_xsize	As 'wp_xsize', but used in 'landscape mode'.
wp_ysize	'Portrait mode': The vertical size of the virtual screen expressed in pixels.
wl_ysize	As 'wp_ysize', but used in 'landscape mode'.
wp_vxorig	'Portrait mode': The x coordinate of the lower left corner of the window (excluding the border) relative to the lower left corner of the virtual screen.
wl_vxorig	As 'wp_vxorig', but used in 'landscape mode'.
wp_vyorig	'Portrait mode': The y coordinate of the lower left corner of the window.
wl_vyorig	As 'wp_vyorig', but used in 'landscape mode'.
wp_vxsize	'Portrait mode': The horizontal size of the window.
wl_vxsize	As 'wp_vxsize', but used in 'landscape mode'.

wp_vysize	Portrait mode : The vertical size of the window.
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 all around. DLBORDER - Double lines border all around. The following principle is used when designating combinations of single ('S') and double ('D') line border types: <left> <right> <upper> <lower> BORD This means that a window limited by double lines all around except for the right side is expressed 'DSDDBORD'.

wp_font	The initial font in 'portrait mode'. The font can be in the range 'A' to 'Z'.
wl_font	As 'wp_font', but used in 'landscape mode'.
w_curfont	The currently used font. **PK förkl
w_level	The level of the window. A newly created window will be on level zero if it is not a special and not a child window, 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 3.29, page 35). The value of this membe significant only if the 'BX_USER' flag in 'w_boxes' is set. This value is assumed to be zero if 'BX_USER' is not set.
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

will be terminated with a signal less than zero.

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

BX_HSCR - The two boxes and the single icon which all three scrolls the text horizontally will be present in the border.

BX_VSCR - As above but scrolls vertically.

BX_CLOS - The 'close' box will 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 four boxes and the two icons which scrolls the text horizontally and vertically will only be 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 3.6 page 18).

BX_HELP - The 'help' box shall be present in the border (see the Winhelp() request 3.30 page 36).

BX_USER - Indicates that the value of the 'w_uboxes' statements is significant.

w_flags	**PMODE**	**- Indicates that coordinates have been given for 'portrait mode'.**
	LMODE	**- As above but for 'landscape mode'.**
	SAVETEXT	**- Save the text contents of the virtual screen.**
	~~**SAVEBITMAP**~~	~~**- Save the bitmap contents of the virtual screen (for future use).**~~
	OVERLAP	**- The window may not be overlapped by another window.**
	LOCK	**- The window is locked on the highest level (level zero).**
	NOOVER	**- The window may not be overlapped by another window.**
	NOCURSOR	- Invisible cursor.

NOMOVE	- The window may not be moved or size changed.
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 levels does not change when the levels 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 3.18, page 27.
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 4.2, page 41).
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' statements are supposed to be given in terms of characters instead of pixels. Note that in this case 'wp_vxorig', 'wl_vxorig', 'wp_vyorig',

upper left corner of the virtual screen.

WGROUP	- This window shall belong to a window group (see 4.2, page 41).

The following four flags have no effect if the actual window is not a child window (see 4.2, page 41). Note that at most one of the four flags may be set.

REL_ULC	- This window shall follow its parent window relative the upper left corner of the parent.
REL_URC	- This window shall follow its parent window relative the upper right corner of the parent.
REL_LLC	- This window shall follow its parent window relative the lower left corner of the parent.
REL_LRC	- This window shall follow its parent window relative the lower right corner of the parent.

w_rstat return status:

W_OK	- everything is well.
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 specified 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 specified window.
WE_FATHER	- the specified 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, (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 statements, 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
wp_font     or   wl_font
```

and

w_color, w_border, w_tsig, w_ntsig.

On exit, the values of these statements 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.

EXAMPLE

The following example shows how to position the lower left corner of the virtual space at the screen coordinates (x,y)= =(350,400). The lower left corner of the window is positioned at the coordinates (x,y)=(0,0) relatively the virtual space. The size of both the window and the virtual screen is (150,200). The addressable area of the window is (0,0) to (149,199). The colour is white, the border is double all around and the font used is B.

The window can be used both in portrait- and landscape modes while the coordinates have been specified for both modes.

The icons 'close box', 'size box', 'move box' and 'zoom box' will be presented in the border.

The text contents of the window will be stored.

```
              .
              .
    struct winstruc win;
              .
              .
              .
    win.wp_xorig = 350;
    win.wp_yorig = 400;
    win.wp_xsize = 150;
    win.wp_ysize = 200;
    win.wp_vxorig = 0;
    win.wp_vyorig = 0;
    win.wp_vxsize = 150;
    win.wp_vysize = 200;
    win.wl_xorig = 350;
    win.wl_yorig = 400;
    win.wl_xsize = 150;
    win.wl_ysize = 200;
    win.wl_vxorig = 0;
    win.wl_vyorig = 0;
    win.wl_vxsize = 150;
```

```
win.w_color = WHITE;
win.w_border = DLBORDER;
win.wp_font = 'B';
win.wl_font = 'B';
win.w_boxes = BX_CLOS ö BX_SIZE ö BX_MOVE Ö BX_ZOOM;
win.w_flags = PMODE ö LMODE ö SAVETEXT;

if (Wincreat(fd, &win) < 0 öö win.w_rstat != W_OK) ä
  printf("Cannot create the window.Ön");
å
          .
          .
          .
```

3.2 MOVE WINDOW TO ZERO LEVEL

The zero level 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
        ä
                sint          l_rstat;
                union
                ä
                        long          l_xxx;
                å l_pad;
        å;
```

'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.

EXAMPLE

The following example moves the window to zero level.

```
        struct winlevel lev;
                  .
                  .
                  .
        if (Winlevel(fd, &lev) < 0 öö lev.l_rstat != W_OK) ä
          printf("Cannot move the window to level zero.Ön");
        å
                  .
                  .
                  .
```

3.3 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)                                              **(one,L)
int                 fd;
struct        winlevel *bp;
```

'fd' is the file descriptor for the window. The 'winlevel' structure is described in section 3.2, page 16.

3.4 ALTER WINDOW

The request for altering parameters of a window (size, position etc) runs

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

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 structure 'winstruc' is described in 3.1, page 9. 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	- everything 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.

EXAMPLE

The following example changes the positions of the lower left corner of the virtual space to the screen coordinat (x,y)= =(200,350). The lower left corner of the window is changed to the home position, (x,y)=(0,0). The sizes of both the virtual space and the window are changed to (350, 100).

```
                .
                .
                .
struct winstruc win;
                .
                .
                .
win.wp_xorig = 500;
win.wp_yorig = 250;
win.wp_xsize = 350;
win.wp_ysize = 100;
win.wp_vxorig = 0;
win.wp_vyorig = 0;
win.wp_vxsize = 350;
win.wp_vysize = 100;

if (Winalter(fd, &win) < 0 öö win.w_rstat != W_OK) ä
  printf("Cannot alter the window.Ön");
å
                .
                .
                .
```

3.5 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 runs

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

3.6 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 previous 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 it was created from. When the default font is changed, the default size and location will remain the same for the newly selected font. The request runs

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

The structure 'winstruc' is described in 3.1, page 9. 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.

[illegible] in the return status

W_OK - everything 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.

EXAMPLE

The following example designates the sizes and locations which are recalled when the icon 'blow up' is activated.

```
         .
            .
            .
 struct winstruc win;
            .
            .
            .
 win.wp_xorig = 100;
 win.wp_yorig = 150;
 win.wp_xsize = 400;
 win.wp_ysize = 400;
 win.wp_vxorig = 0;
 win.wp_vyorig = 0;
 win.wp_vxsize = 400;
 win.wp_vysize = 400;

 if (Windflsz(fd, &win) < 0 öö win.w_rstat != W_OK) ä
   printf("Cannot set up default size and location for the window.Ön
 å
            .
            .
            .
```

3.7 ALTER WINDOW FLAGS

The request for altering the flags in the 'w_flags'' unit descriptor 'word' for a window runs

```
        Winflags(fd, bp);
        int                   fd;
        struct         flgstruc *bp;
```

The structure 'flgstruc' looks like:

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

        struct         flgstruc
        ä
                uflags         f_flags;
                sint         f_rstat;
                union
                ä
                        long         f_xxx;
                å f_pad;
          å;
```

'f_flags' is the new flags for the window.

Contains some flags of which the following may be altered:

```
                LOCK        NOOVER      NOCURSOR    NOMOVE
                ALLSCR      KEYSCROLL   WRITSCROLL  NOCPIN
                NOCPOUT     REL_ULC     REL_URC     REL_LLC
                REL_LRC.
```

The following flags are ignored:

```
                PMODE       LMODE       SAVETEXT    SAVEBITMAP
                OVERLAP     SPECIAL     ALTMPNT     RELATIVE
                TXTSIZE     WGROUP
```

(These flags are explained in page 13.)

'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 is set.

EXAMPLE

The following example removes the cursor from the window and prevents the size and location of the window to be changed.

```
                .
                .
                .
struct flgstruc flg;
                .
                .
                .
flg.f_flags = NOCURSOR ö NOMOVE;

if (Winflags(fd, &flg) < 0 öö flg.f_rstat != W_OK) ä
  printf("Cannot alter the window flags.Ön");
å
                .
                .
                .
```

3.8 GET WINDOW STATUS

The request for getting the current status of a window runs

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

The structure 'winstruc' was described in 3.1, page 9. 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. The 'PMODE' and 'LMODE' flags indicate which one it is.

'w_rstat' is the return status

    W_OK   - everything is well.

    WE_NOTCR   - the window has not been created yet.

3.9 INSERT A HEADER IN A WINDOW BORDER

The request for inserting a header (for example the program name) in the border of a window runs

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

where the structure 'headstruc' looks like:

```
        typedef         unsigned short          word;

        struct          headstruc
        ä
                char            h_hdrÄHDRSIZEÅ;
                word            h_flags;
                union
                ä
                        long            h_xxx;
                å h_pad;
        å;
```

'h_hdrÄÅ' is the header string. 'h_flags' contains the flags:

    H_INVHD - Invert the window header (relatively the window background).

    H_INVTOP - Invert the top window header (relatively 'H_INVHD').

Note that the header can be created before the window is created.

EXAMPLE

The following example writes an inverted header into the window border The header used is ' FIRSTLINE' .

```
                    .

                    .
          struct headstruc head;

                    .

                    .


          strcpy(head.h_hdr, " FIRSTLINE ");
          head.h_flags = H_INVHD;

          if (Winheader(fd, &head) < 0) ä
            printf("Cannot insert a header in the window border.Ön");
          å
                    .

                    .

                    .
```

3.10 ICON SUPPORT

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

The request runs

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

and is used to specify that when pointing 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 structure 'winicon' looks like:

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

        struct         winicon
        ä
                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_cmdseqÄICONSEQLENÅ;
                word         i_flags;
                sint         i_rstat;
                union
                ä
                        long         i_xxx;
                å  i_pad;
        å;
```

'ip_xorig' 'ip_yorig' 'il_xorig' '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' 'ip_ysize' 'il_xsize' 'il_ysize'	is the width and height of the area in 'portrait' and 'landscape mode', respectively.
'i_cmdseqÄÅ'	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

I_PMODE	- Portrait mode coordinates are given.
I_LMODE	- Landscape mode coordinates are given.
I_PRESS	- Send the sequence when pointing to the area and the left button is pressed.

I_RELEASE - Send the sequence when pointing to the area and the left button is released.

I_INVERT - Invert the area occupied by the icon when 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 leaving 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 either one or both of 'I_ENTER' and 'I_LEAVE' is set, it is checked whether the mouse pointer is inside or outside (respectively) the specified area. If it is, 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 terms 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 relatively 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 adjustec

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'.

'i_rstat' is the return status:

W_OK - everything is well.

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.

EXAMPLE

The following example positions the icon in the lower left
corner of a virtual space. The size of the icon is (x,y)=
=(150,75).

The icon can be used in both portrait- and landscape modes
while coordinates have been specified for both modes.

The icon will send a sequence when the left key is pressed or
released. The icon is inverted when pointed at.

```
          .
          .
          .
struct winicon icon;
          .
          .
          .
icon.ip_xorig = 0;
icon.ip_yorig = 0;
icon.ip_xsize = 150;
icon.ip_ysize = 75;
icon.il_xorig = 0;
icon.il_yorig = 0;
icon.il_xsize = 150;
icon.il_ysize = 75;
strcpy(icon.i_cmdseq, "AAA");
icon.i_flags = I_PMODE ö I_LMODE ö I_PRESS ö I_RELEASE ö I_INVERT;

if (Winicon(fd, &icon) < 0 öö icon.i_rstat != W_OK) ä
  printf("Cannot create the icon.Ön);
å
          .
          .
```

3.11 REMOVE ICONS

The request for removing all set up icons for a window runs

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

EXAMPLE

The following example removes all the icons for the window.

```
                      .
                      .
                      .
          if (Rmicons(fd) < 0) ä
            printf("Cannot remove all icons for the window.Ön");
          å
                      .
                      .
                      .
```

3.12 MOUSE SUBSTITUTING 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 keys or other special keys on the ABC99 keyboard (these keys generates codes with the most significant bit set).

The request for specifying these keys runs

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

The structure 'substit' looks like:

```
        typedef        char        sint;

        struct        substit
```

```
                    unsigned char         c_keysÄSUBSTKEYSÅ;
                    unsigned char         c_step;
                    unsigned char         c_lstep;
                    union
                    ä
                            long        c_xxx;
                    å c_pad;
           å;
```

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.

KEYWORD	- DESCRIPTION
c_initflg	If 'ON', the mouse simulation keys will be enabled after this request. If 'OFF' they will initially be disabled.
c_keysÄÅ	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).

c_keysÄÅ The array consists of the following keys:

S_ONOFF	The key used to define whether the mouse shall be active or inactive.
S_MPU	Move mouse pointer up.
S_MPD	down.
S_MPL	left.
S_MPR	right.
S_MPUL	up - left.
S_MPUR	up - right.
S_MPDL	down - left.
S_MPDR	down - right.
S_LMPU	up long.
S_LMPD	down long.
S_LMPL	left long.
S_LMPR	right long.
S_LMPUL	up - left long.
S_LMPUR	up - right long.
S_LMPDL	down - left long.
S_LMPDR	down - right long.
S_PCMD	Point to command key (replaces the left key of the mouse).
S_CHWIN	Change window level key (replaces the right key of the mouse).
S_MCA	Mark text area to copy (replaces the middle key of the mouse).

3.13 ALTER THE BACKGROUND PATTERN

The request for altering the pattern of the background runs

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

The structure chbgstruc looks like:

```
typedef         unsigned short         word;

struct          chbgstruc
ä
        word            cb_bitmapÄBGPSIZEÅ;
        union
        ä
                long            cb_xxx;
        å cb_pad;
å;
```

'cb_bitmapÄÅ' 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 unit descriptor 'word' is displayed to the left on the screen.

EXAMPLE

The following example changes the background pattern to a stilistic 'ABC1600'.

```
#include <win/w_const.h>
#include <win/w_types.h>
#include <win/w_structs.h>
#include <win/w_macros.h>

struct chbgstruc chbg =
ä
  0x0, 0x338e, 0x4a50, 0x4a50,
  0x7b90, 0x4a50, 0x4a50, 0x4b8e,
  0x0, 0x4c44, 0x50aa, 0x54aa,
  0x5aaa, 0x52aa, 0x4c44, 0x0
å;

main()
ä
  int fd;

  if (fd = open("/win/activate", 2)) == -1) ä
    printf("Cannot open the window.Ön");
  å

  if (Winchbg(fd, &chbg) < 0) ä
    printf("Cannot alter the background pattern.Ön");
  å

  sleep(15);

  close(fd);
å
```

3.14 GET THE VISIBLE PARTS OF A WINDOW OR THE BACKGROUND

The request for getting the visible parts of a window or the background runs

```
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 structure 'buffer', and looks like:

```
struct        buffer
ä
        struct        visdes        v;
        struct        rectdes       bÄVSIZEÅ;
å;
```

The structure 'visdes' is a parameter structure and looks like:

```
typedef        char        sint;

struct        visdes
ä
        short        v_nrect;
        sint        v_rstat;
        union
        ä
                long        v_xxx;
        å v_pad;
å;
```

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

```
typedef        short        pix_d;

struct        rectdes
ä
        pix_d        r_xorig;
        pix_d        r_yorig;
        pix_d        r_xsize;
```

a;

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).

3.15 INVERSE VIDEO

The request runs

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

and changes the screen to inverse video.

EXAMPLE

The following example changes the character contents of the entire screen to inverse video.

```
          #include <win/w_const.h>
          #include <win/w_types.h>
          #include <win/w_structs.h>
          #include <win/w_macros.h>

          main()
          if (fd = open("/win/activate", 2)) == -1) ä
            printf("Cannot open the window.Ön");
          å

          if (Winivideo(fd) < 0) ä
            printf("Cannot inverse video.Ön");
          å

          sleep(15);

          close(fd);
        å
```

3.16 NORMAL VIDEO

The request runs

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

and restores the screen to normal video.

3.17 MAKE THE CURSOR VISIBLE IN THE WINDOW

The request for making the cursor visible in the window runs

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

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

EXAMPLE

The following example makes the cursor visible.

```
                  .
                  .
                  .
          if (Wincurvis(fd) < 0) ä
            printf("Cannot make the cursor visible in the window.Ön");
          å
                  .
```

3.18 CHANGE MOUSEPOINTER

The request for changing the layout of the mousepointer runs

```
        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
        ä
                pix_d         np_xsize;
                pix_d         np_ysize;
                pix_d         np_xpnt;
                pix_d         np_ypnt;
                dword         np_andÄMPSIZEÅ;
                dword         np_orÄMPSIZEÅ;
                byte        np_flags;
                sint        np_rstat;
                union
                ä
                        long         np_xxx;
                å np_pad;
        å;
```

'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 mousepointer. It shall be specified relative the upper left corner of the mousepointer.

Each pixel row of the mouse pointer is constructed by the operation

```
        (x & np_andÄprowÅ) ö np_orÄprowÅ
```

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.

3.18 CHANGE MOUSEPOINTER

The request for changing the layout of the mousepointer runs

```
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
ä
        pix_d        np_xsize;
        pix_d        np_ysize;
        pix_d        np_xpnt;
        pix_d        np_ypnt;
        dword        np_andÄMPSIZEÅ;
        dword        np_orÄMPSIZEÅ;
        byte       np_flags;
        sint       np_rstat;
        union
        ä
                long        np_xxx;
        å np_pad;
å;
```

'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 mousepointer. It shall be specified relative the upper left 'np_andÄÅ' and 'np_orÄÅ' are masks used to construct the mousepointer.

Each pixel row of the mouse pointer is constructed by the operation

```
        (x & np_andÄprowÅ) ö np_orÄprowÅ
```

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 mousepointer can not be changed.

3.19 GET NUMBER OF OPEN WINDOWS

The request for finding out how many windows which are open or created runs

```
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
ä
        short           nw_open;
        short           nw_created;
        union
        ä
                long            nw_xxx;
        å nw_pad;
å;
```

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

3.20 RESTORE SCREEN

The request for restoring the screen (i e rewriting the whole screen) runs

```
int         fd;
```

EXAMPLE

The following example restores the screen.

```
#include <win/w_const.h>
#include <win/w_types.h>
#include <win/w_structs.h>
#include <win/w_macros.h>

main()
ä
  int fd;

  if (fd = open("/win/activate", 2)) == -1) ä
    printf("Cannot open the window.Ön");
  å

  if (Winrestor(fd) < 0) ä
    printf("Cannot restore screen.Ön");
  å

  sleep(15);

  close(fd);
å
```

3.21 GET TEXT CONTENTS OF A WINDOW

The request for getting the text contents of a window runs

```
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.

The structure 'buffer' looks like:

```
struct      buffer
ä
        struct      txtstruc s;
        char                bÄBSIZEÅ;
å;
```

The structure 'txtstruc' looks like:

```
typedef      short      cur_d;
typedef      char      sint;

struct      txtstruc
ä
        cur_d       tx_row;
        cur_d       tx_col;
        cur_d       tx_rcnt;
        cur_d       tx_ccnt;
        sint       tx_rstat;
        union
        ä
                long        tx_xxx;
        å tx_pad;
å;
```

'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)

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.

3.22 TEST IF THE WINDOW HANDLER IS ACTIVATED

The request for testing if the window handler is activated runs

```
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.

EXAMPLE

The following example checks if the window handler is activated.

```
            .
                .
                .
 if (Wintest(fd) < 0) ä
   printf("The window handler is not activated.Ön");
 å
                .
                .
                .
```

3.23 SET INITIAL DRIVER AND TERMINAL PARAMETERS

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

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

The structure 'wininit' looks like:

```
typedef          unsigned long          t_stop;
```

```
struct          wininit
ä
        t_stop          td_tbstopÄTSTOPSIZEÅ;
        word          td_term;
        struct
        ä
                unsigned short          c_iflag;
                unsigned short          c_oflag;
                unsigned short          c_cflag;
                unsigned short          c_lflag;
                char                    c_line;
                unsigned char           c_ccsÄ8Å;
        å td_driver;
        union
        ä
                long          td_xxx;
        å td_pad;
å;
```

'td_tbstopÄÅ' 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 VT100 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 structure 'termio' (see the header file <sys/termio.h> and the documentation for the ioctl() unix system call).

The default tab stops are places every eighth position. Of the terminal flags, the 'TD_WRAP' flag is set by default. The driver parameters are the same as those of the console when the window handler was activated.

3.24 GET INITIAL DRIVER AND TERMINAL PARAMETERS

The request for getting the values of the initial driver and terminal parameters runs

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

3.25 SET UP A ZOOM LIST FOR A WINDOW

A zoom list is a list of fonts to change between when pointing 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 structure 'zoomlst'.

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

The request for setting up a zoom list runs

```
        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
        ä
                char         zp_listÄZOOMSIZEÅ;
                char         zl_listÄZOOMSIZEÅ;
                byte         z_flags;
                sint         z_rstat;
                union
                ä
                        long         z_xxx;
                å z_pad;
        å;
```

'zp_listÄÅ' is the list of fonts to be used in portrait mode and 'zl_listÄÅ' is used in landscape mode.

'z_flags' contains some flags:

Z_PMODE - 'Portrait mode' zoom list is given.

Z_PMODE

Z_LMODE - 'Landscape mode' zoom list is given.

'z_rstat' is the return status:

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

EXAMPLE

The following example shows a zoomlist which makes toggling between the default font of the window and the fonts I, L and R possible.

```
                .
                .
                .
     struct zoomlst zoom;
                .
                .
                .
     strcpy(zoom.zp_list, "ILR");
     strcpy(zoom.zl_list, "ILR");
     zoom.z_flags = Z_PMODE ö Z_LMODE;

     if (Winzoom(fd, &zoom) < 0 öö zoom.z_rstat != W_OK) ä
       printf("Cannot set up a zoom list for the window.Ön");
     å
                .
                .
                .
```

3.26 CHANGE THE DEFAULT FONT FOR A WINDOW

The request for changing the default font for a window runs

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

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

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

        struct         dfltchr
        ä
                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
                ä
                        long         dc_xxx;
                å dc_pad;
        å;
```

'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'.

----

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 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. However, the size of the virtual screen is adjusted so it contains the same number of character rows and columns.

EXAMPLE

The following example sets the default font for the window to D.

```
          .
          .
          .
struct dfltchr dflt;
          .
          .
          .
dflt.dcp_font = 'D';
dflt.dcl_font = 'D';
dflt.dc_flags = Z_PMODE ö Z_LMODE;

if (Winndchr(fd, &dflt) < 0 öö dflt.dc_rstat != W_OK) ä
  printf("Cannot change the default font for the window.Ön");
å
          .
          .
          .
```

3.27 TURN THE SCREEN

The request for turning the screen from portrait to landscape mode or vice versa runs

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

All channels, except the one obtained when the window handler was activated, must be closed.

The structure 'modstruc' looks like:

```
typedef         char            sint;

struct          modstruc
ä
        sint            m_mode;
        sint            m_rstat;
        union
        ä
                long            m_xxx;
        å m_pad;
å;
```

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

'm_rstat' is the return status:

W_OK   - everything is ok.

WE_OPEN - there are windows open.

3.28 GET SCREEN MODE

The request for getting the current screen mode ('portrait' or 'landscape') runs

```
        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 structure 'modstruc' is described in section 3.27, page 34. The 'm_mode' statement contains the curre mode ('M_PORT' or 'M_LAND') and 'm_rstat' is always 'W_OK'.

3.29 ADD AN USER DEFINED BOX

In the left side of the window border there are user defined boxes of 16x16 pixels. 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 3.1, page 9).

The request for setting up a user defined box runs

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

'fd' is the file descriptor for the window.

The structure 'userbox' looks like:

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

        struct          userbox
        ä
                word        bx_bmapÄUBOXSIZEÅ;
                short        bx_sig;
                byte        bx_flags;
                sint        bx_rstat;
                union
                ä
                        long        bx_xxx;
                å bx_pad;
        å;
```

bx_bmapÄÅ	contains the bitmap for the box. Note that the most significant bit in a unit descriptor '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.

3.30 ALTER HELP BOX SEQUENCE

The help box is a box in the upper side of the border containing a question mark which upon use puts a character sequence on the keyboard input buffer. The intention is that all programs use this facility so that help can be requested in a similar manner in all programs.

When a window is opened, the help box sequence is initialized to a '?' (question mark). The request for altering this to another sequence runs

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

'fd' is the file descriptor for the window.

The structure 'helpst' looks like:

```
typedef        unsigned short        word;

struct        helpst
ä
        char        hlp_seqÄHLPSIZEÅ;
        word        hlp_flags;
        union
        ä
                long        hlp_xxx;
        å hlp_pad;
å;
```

'hlp_seqÄÅ' 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.

3.31 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. Reading the keyboard buffer will not lead to wait.

The request runs

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

The structure 'kysigst' looks like:

```
        struct        kysigst
        ä
                sint        ks_sig;
                byte        ks_flags;
                sint        ks_rstat;
                union
                ä
                        long        ks_xxx;
                å ks_pad;
        å;
```

'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 versions.

'ks_rstat' is the return status:

```
        W_OK            - everything is well.

        WE_ILPARA       - an illegal signal was specifiead.
```

3.32 READ THE CONTENTS OF THE PICTURE MEMORY

The request for reading the contents of the picture memory for a window or the whole screen runs

```
        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 structure 'buffer' consists of a parameter structure followed by a buffer. The buffer is big enough to hold the contents of the specified picture memory area and looks like:

```
        typedef         unsigned char         byte;

        struct        buffer
        ä
                struct        wpictblk p;
                byte                   bÄBSIZEÅ;
        å;
```

The structure 'wpictblk' looks like:

```
        typedef         short         pix_d;

        struct        wpictblk
        ä
                pix_d         p_xaddr;
                pix_d         p_yaddr;
                pix_d         p_width;
                pix_d         p_height;
                union
                ä
                        long         p_xxx;
                å p_pad;
        å;
```

'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 '.bÄÅ' corresponding to non-visible areas of a virtual screen will contain zeros (i e cleared bits).

Note that the most significant bit in a byte is displayed to the left on the screen. (While all pixels are single checked, the execution of this request is time demanding).

3.33 ALTER THE SPRAY MASK

This request changes the pattern of 32 times 32 pixels used by the 'spray' escape sequence (see the document in swedish; ANVÄNDARHANDLEDNING ABC1600 FÖNSTERHANTERARE).

The request runs

```
        Spraymask(fd, bp)
        int                fd;
        struct        sprayst        *bp;
```

'fd' is the file descriptor for the window.

The structure 'sprayst' looks like:

```
        typedef         unsigned long           dword;

        struct          sprayst
        ä
                dword           sp_maskÄ8*sizeof(dword)Å;
        å;
```

where 'sp_maskÄÅ' 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.

4 MISCELLANEOUS

4.1 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. 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.

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	As above but reading the function keys.
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.
TIOCSETP	Flush and then set the basic parameters.
TIOCSETN	Set the basic parameters (no flush).
TIOCEXCL	Set 'exclusive-use mode'.
TIOCNXCL	Turn off 'exclusive-use mode'.
TIOCFLUSH	Flush input and output queues.
TIOCSETC	Set the special characters.
TIOCGETC	Get the special characters.
FIORDCHK	Check if any character is input.
TCSETAF	Wait for output to drain, then flush the input queues and set the parameters for the terminal.
TCSETAW	As above, but does 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.

4.2 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.

4.3 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:

1) The escape sequence   'ESC : <n> H'   is sent to the window

2) 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. (While method 1) has   **PK specify the
some side   effects!
effects, method 2) is mostly prefered).

1) Send the 'Reset to Initial State' escape sequence ('ESC c') to the window

2) Send the   'ESC : J'   escape sequence to the window when the current font is the same as the default font for the window.

4.4 AUTOMATICALLY SUPPORTED FUNCTIONS

THE MOUSE

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

When pointing to a region marked by the 'Winicon()' request, the area is inverted if the 'I_INVERT' flag is set. Then if the left button on the mouse is pressed, the specified code sequence is sent to the appropriate process.

CHANGE WINDOW SIZE

When pointing 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.

MOVE WINDOW AND VIRTUAL SCREEN

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.

SELECT THE VISIBLE PART OF THE VIRTUAL SCREEN

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. Then press the left button and move 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.

COPY TEXT BETWEEN WINDOWS

To copy a region (a rectangle) of text from one window to another, put the pointer at the upper left character of the rectangle. Then 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 cancel the operation, press any button (except the middle one).
An alternative is to move the pointer to the destination window and press the middle button once more, causing the marked region to be copied.

Note, that since the text contents of all the windows are stored by the window handler, this operation will also work with programs not knowing about the windows.

CHANGE TOP LEVEL WINDOW

To change a window as 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 is 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.

5 EXAMPLES

The three following program examples will help you to get familiar to the use of windows of different types. Example one opens a window in both 'portrait' and 'landscape mode', ex two shows the use of icons, and number three emulates a specific video terminal.

EXAMPLE 1

This program creates a window which can be used in both 'portrait' and 'landscape mode'.

The lower left corner of the window is placed at (x,y)=(350,400) and the size of the window is 150 * 150 pixels. The coordinates for the lower left corner is (0,0) and the size in the window is 150 * 150 pixels. The colour of the window is white, the font used is 'A', and no border is used.

```
#include    <win/w_const.h>
#include    <win/w_types.h>
#include    <win/w_structs.h>
#include    <win/w_macros.h>

/*
 *  The structure 'winstruc' is used for creating windows.
 */
struct winstruc win;

main()
ä
    int    fd;
    char    c;

    /*
     *  Fill in the structure.
     */
    win.wp_xorig = 350;
    win.wl_xorig = 350;
    win.wp_yorig = 400;
    win.wl_yorig = 400;
    win.wp_xsize = 150;
    win.wl_xsize = 150;
    win.wp_ysize = 150;
    win.wl_ysize = 150;
    win.wp_vxorig = 0;
    win.wl_vxorig = 0;
    win.wp_vyorig = 0;
    win.wl_vyorig = 0;
    win.wp_vxsize = 150;
    win.wl_vxsize = 150;
    win.wp_vysize = 150;
    win.wl_vysize = 150;
    win.w_color = WHITE;
    win.w_border = NOBORDER;
    win.wp_font = 'A';
    win.wl_font = 'A';
    win.w_flags = PMODE ö LMODE;

    /*
     *  Open a channel for the window.
     */
    if ((fd = open("/win", 2)) == -1) ä
      printf("Cannot open a channel for the window.Ön");
    å
    /*
     *  Create the window.
     */
    if (Wincreat(fd, &win) < 0 öö win.w_rstat != W_OK) ä
      printf("Cannot create the window.Ön");
    å

    /*
     *  Write text in the window.
     */
    write(fd, "Press the 'RETURN' key.", 15);

    /*
     *  Wait for the 'RETURN' key to be pressed.
     */
    read(fd, &c, 1);

    /*
     *  Delete the window.
     */
    close(fd);
å
```

EXAMPLE 2

The following program creates a window which has two icons. The icons turns inverted when pointing at them.

The window has no cursor.

The first icon sends an 'A' when pointing at the icon and the left mouse key pressed. The second icon sends a 'B' when pointing at it and the left key is released.

The lower left corner of the window is placed at (x,y)=(0,395) and the size of the window is (x*y)=(768*84).

The coordinate of the lower left window corner is (0,0) and the size in the window is (x*y)=(768*84).

The colour of the window is black, the font used is 'A', and no border is used.

The icon no 1 is placed at (x,y)=(133,5) relatively the lower left corner of the window. The size of the icon is (x*y)=(120*75)

The icon no 2 is placed at (x,y)=(514,5) relatively the lower left corner of the window. The size of the icon is (x*y)=(120*75)

```
#include    <win/w_const.h>
#include    <win/w_types.h>
#include    <win/w_structs.h>
#include    <win/w_macros.h>

/*
 *  The structure 'winstruc' is used for creating windows.
 */
struct winstruc win;

/*
 *  The structure 'winicon' is used for creating icons.
 */
[illegible]
```

```
main()
ä
    int     fd;
    char     c;

    /*
     *  Fill in the structure for the window.
     */
    win.wp_xorig = 0;
    win.wp_yorig = 395;
    win.wp_xsize = 768;
    win.wp_ysize = 84;
    win.wp_vxorig = 0;
    win.wp_vyorig = 0;
    win.wp_vxsize = 768;
    win.wp_vysize = 84;
    win.w_color = BLACK;
    win.w_border = NOBORDER;
    win.wp_font = 'A';
    win.w_flags = PMODE ö NOCURSOR;

    /*
     *  Open a channel for the window.
     */
    if ((fd = open("/win", 2)) == -1) ä
      printf("Cannot open a channel for the window.Ön");
    å

    /*
     *  Create the window.
     */
    if (Wincreat(fd, &win) <0 öö win.w_rstat != W_OK) ä
      printf("Cannot create the window.Ön");
    å

    /*
     *  Fill in the structure for icon number one.
     */
    icon.ip_xorig = 133;
    icon.ip_yorig = 5;
    icon.ip_xsize = 120;
    icon.ip_ysize = 75;
    strcpy(icon.i_cmdseq,"A");
    icon.i_flags = I_PMODE ö I_PRESS ö I_INVERT;

    /*
     *  Create icon number one.
     */
    if (Winicon(fd, &icon) <0 öö icon.i_rstat != W_OK) ä
      printf("Cannot create icon number one.Ön");
    å

    /*
     *  Fill in the structure for icon number two.
     */
    icon.ip_xorig = 514;
    icon.ip_yorig = 5;
    icon.ip_xsize = 120;
    icon.ip_ysize = 75;
    strcpy(icon.i_cmdseq,"B");
    icon.i_flags = I_PMODE ö I_RELEASE ö I_INVERT;
    /*
     *  Create icon number two.
     */
    if (Winicon(fd, &icon) <0 öö icon.i_rstat != W_OK) ä
      printf("Cannot create icon number two.Ön");
    å

    /*
     *  Wait for the 'RETURN' key to be pressed.
     */
    read(fd, &c, 1);

    /*
     *  Delete the window.
     */
    close(fd);
å
```

EXAMPLE 3

This function creates and opens a window running in the program specified. The window will have the status of a terminal with the same standard input, output, and error output as the window.

A specified header will be inserted in the window header if the pointer to this structure is not NULL.

Before the program is executed, the current directory will be changed to the one specified if the pointer to this structure is not NULL.

This function 'fork' does not wait for the process to terminate. Before the execution of the program, all files will be closed, except those with file descriptors zero, one, and two.

The signal set up is unaffected, therefore signals already ignored will remain ignored when the program is executed. No errors are returned, instead the child process will be terminated if an error occurs.

If the execution of the program fails, an error message is displayed in the new window. (No message is displayed if the pointer to the message is NULL.

```
        #include        <stdio.h>
        #include        <fontl.h>
        #include        "../wincl/w_const.h"
        #include        "../wincl/w_types.h"
        #include        "../wincl/w_structs.h"
        #include        "../wincl/w_macros.h"

        w_term(wdp, hdp, dir, name, argv, errp)
        struct winstruc   *wdp;              /* pointer to window data
        struct headstruc  *hdp;       /* pointer to window data
                                         header data or NULL
        char                   *dir;             /* directory to change to o
        char                   *name;     /* name of the file to execute
        char                   **argv;    /* program arguments
        char                   *errp;     /* error message to display if the
                                         execution fails
        ä
                register        pid;  /* process id
                register        s;    /* returned status
                int                r;
```

```
/* Start the program. Note that two 'fork' are necessary
 * to avoid processes not waited for. In this way the
 * init process will wait for the double forked process.
 */

if ((pid = fork())  ! =0) ä
        while ((s=wait(&r)) ! = pid && s != -1)
                ;
        return;
å
if (fork() !=0) ä
        exit(0);
å

/*
 * Close all open files.
 */
for (s = 0 ; s < _NFILE ; s ++)  ä
        close(s);
å

/*
 * Set up new process group (equal to the process id).
 */
if (setpgrp()) < 0 öö

/*
 * Open a channel for the new window.
 */

        open(WMNTDIR, 2) ! = 0 öö

/*
 * Set up a new controlling terminal.
 */
        fontl(0, F_SETCT, 0) < 0 öö
          c
/*
 * Do two 'dup' to create the standard and error outputs.
 */
        dup(0) !=1 öö dup(0) != 2 öö

/*
 * Create the window.
 */
        Wincreat(0, wdp) <0 öö wdp->w_rstat !=W_OK öö

/*
 * Insert possible header to the window.
 */
        (hdp != NULL && Winheader(0, hdp) < 0) öö

/*
 * Possibly change to another directory.
 */
        (dir != NULL && chdir(dir) < 0)) ä
                _exit(1);
å
```

```
        /*
         * Execute the desired program.
         */
        execv(name, argv);
        if (errp != NULL) ä
                write(0, errp, strlen(errp));
                sleep(5);
        å

        _exit(1);
å
```

6 WINDOW UTILITY COMMANDS

In the following syntax description the characters 'Ä' and 'Å' marks statements or words that may be omitted. ⟨n⟩ symbolizes a numerical value.

6.1 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 Ä-bwnotzÅ Ä-c <n>Å Ä-r <n>Å Ä-h <n>Å Ä-w <n>Å Ä-x <n>Å
              Ä-y <n>Å Ä-f <c>Å Ä-s <n>Å Ä-e <n>Å Ä<command>Å
```

OPTION - DESCRIPTION

b - Black window.
w - White window (default).
n - No window border.

o - Single (one) line window border.
t - Double (two) lines window border (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 indicate that an operation on the window has been completed. (default zero).
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 zero).

6.2 WHEAD

This command inserts a header in a window. If no header is given, the present header will be removed.

The syntax is

```
        whead Ä-iÅ Ä-tÅ Ä<header>Å
```

OPTION - DESCRIPTION

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

6.3 WICON

This command sets up an icon in a window.

The syntax is

```
        wicon Ä-prielmqsztÅ Ä-x <n>Å Ä-y <n>Å Ä-w <n>Å Ä-h <n>Å
              Ä<sequence>Å
```

OPTION - DESCRIPTION

p - 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 pointing to it.

e - Send the icon sequence when entering the icon area.
l - Send the icon sequence when leaving 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' (edward) or 'l' (london) 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 zero).
y - The y coordinate of the lower left corner of the icon
    (default zero).

w - The width of the icon (default 100).
h - The height of the icon (default 100).
<sequence> - the icon sequence to be sent when the icon is chosen.

6.4 RMICONS

This command removes all icons in a window.

The syntax is

```
        rmicons
```

6.5 WZOOM

This command sets up a zoom list for a window.

The syntax is

```
        wzoom Ä<zoomlist>Å
```

<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.

6.6 WFONT

This command changes the default font for a window.

The syntax is

```
        wfont Ä-x <n>Å Ä-y <n>Å Ä<font>Å
```

OPTION - DESCRIPTION

x - The x coordinate for the middle visible character (default one).
y - The y coordinate for the middle visible character (default one).
<font> - 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.

6.7 WTOP

This command moves a window to the top level.

The syntax is

```
        wtop
```

6.8 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 Ä-nÅ Ä<file>Å
```

the option '-n' shall be used if no error messages shall be displayed. If <file> is not specified, the standard input is read instead.

6.9 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 Ä-nÅ Ä<file>Å
```

the option '-n' shall be used if no error messages shall be displayed. If <file> is not specified, the standard input is read instead.

6.10 WMP

This command reads the file specified as argument and uses the data 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 Ä-nÅ Ä<file>Å
```

the option '-n' shall be used if no error messages shall be displayed. If <file> is not specified, the standard input is read instead.

6.11 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 Ä-nÅ Ä<file>Å
```

the option '-n' shall be used if no error messages shall be displayed. If <file> is not specified, the standard input is read instead.

6.12 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 Ä<infile>Å Ä-o <outfile>Å
```

Where <infile> is the input file (default '.window') and <outfile> is the output file (default standard output). **3.32

6.13 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 structure 'wpictblk' (see).

The syntax is

```
        wpictrd Ä-pÅ Ä-x <n>Å Ä-y <n>Å Ä-w <n>Å Ä-h <n>Å Ä-c <n>Å
                Ä-o <file>Å
```

OPTION - DESCRIPTION

p - first output a header parameter.
x - x pixel coordinate of the lower left corner of the rectangle to read (default zero).
y - y pixel coordinate of the lower left corner of the rectangle to read (default zero).

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 zero, i e standard input).

o - The name of the output file. If not specified, the output is written to the standard output.

6.14 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 Ä-tÅ Ä-x <n>Å Ä-y <n>Å Ä-u <n>Å Ä-v <n>Å Ä-w <n>Å
              Ä-h <n>Å
```

OPTION - DESCRIPTION

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.

6.15 WHELP

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

The syntax is

```
        whelp Ä<sequence>Å
```

No sequence will be sent if <sequence> is not given.

7 INDEX

a
**** not completed ****

production: IFS DATA AB Linköping and Luxor Datorer AB Motala