Contents

1 Module `Curses` : Bindings to the ncurses library. 1

1.1 Initialization functions . . . 1
1.2 Cursor . . . 2
1.3 Operations on characters . . . 2
1.3.1 Displaying characters . . . 4
1.3.2 Attributes . . . 6
1.3.3 Background . . . 10
1.3.4 Operations on lines . . . 10
1.3.5 Characters input . . . 10
1.4 Windows . . . 14
1.4.1 Window manipulations . . . 14
1.4.2 Refresh control . . . 15
1.4.3 Overlapped windows . . . 15
1.4.4 Decorations . . . 16
1.4.5 Pads . . . 16
1.5 Colors . . . 17
1.6 Input/output options . . . 17
1.6.1 Input options . . . 17
1.6.2 Output options . . . 18
1.7 Soft-label keys . . . 18
1.8 Mouse . . . 19
1.9 Misc . . . 19
1.10 Screen manipulation . . . 19
1.11 Terminal . . . 20
1.12 Low-level curses routines . . . 21
1.13 Configuration . . . 21

1 Module `Curses` : Bindings to the ncurses library.

Beware, all coordinates are passed `y` first, then `x`.

Functions whose name start with a "w" take as first argument the window the function applies to. Functions whose name start with "mv" take as first two arguments the coordinates `y` and `x` of the point to move the cursor to. For example `mvaddch y x ch` is the same as `move y x; addch ch`.

`type window`

Windows.

`type screen`

Screens.

`type terminal`

Terminals.

`type chtype = int`

Characters. Usual characters can be converted from/to `chtype` using `char_of_int` and `int_of_char`. See also `get_acs_codes` for characters useful for drawing and the `Key` module for special input characters.

`type attr_t = int`

Attributes are `lor`ings of flags which are defined in the `A` module.

`type err = bool`

A return value. `false` means that an error occured.

1.1 Initialization functions

`val initscr : unit -> window`

Initialize the curses library.

`val endwin : unit -> unit`

Restore the terminal (should be called before exiting).

`val isendwin : unit -> bool`

Has `endwin` been called without any subsequent call to `werefresh`?

`val newterm : string -> Unix.file_descr -> Unix.file_descr -> screen`

Create a new terminal.

`val set_term : screen -> unit`

Switch terminal.

`val delscreen : screen -> unit`

Delete a screen.

`val stdscr : unit -> window`

1.2 Cursor

`val getyx : window -> int * int`

Get the current cursor position.

`val getparyx : window -> int * int`
`val getbegyx : window -> int * int`
`val getmaxyx : window -> int * int`
`val move : int -> int -> err`

Move the cursor.

`val wmove : window -> int -> int -> err`

1.3 Operations on characters

```
module Acs :
  sig

     type acs = {
       ulcorner : Curses.chtype ;
```

Upper left-hand corner (+).

```
       llcorner : Curses.chtype ;
```

Lower left-hand corner (+).

```
       urcorner : Curses.chtype ;
```

Upper right-hand corner (+).

```
       lrcorner : Curses.chtype ;
```

Lower right-hand corner (+).

```
       ltee : Curses.chtype ;
```

Left tee (+).

```
       rtee : Curses.chtype ;
```

Tight tee (+).

```
       btee : Curses.chtype ;
       ttee : Curses.chtype ;
       hline : Curses.chtype ;
```

Horizontal line (-).

```
       vline : Curses.chtype ;
```

Vertical line (|).

```
       plus : Curses.chtype ;
```

Plus (+).

```
       s1 : Curses.chtype ;
```

Scan line 1 (-).

```
       s9 : Curses.chtype ;
```

Scan line 9 (_).

```
       diamond : Curses.chtype ;
```

Diamond (+).

```
       ckboard : Curses.chtype ;
       degree : Curses.chtype ;
```

Degree symbol (').

```
  plminus : Curses.chtype ;
```

Plus/minus (#).

```
  bullet : Curses.chtype ;
  larrow : Curses.chtype ;
```

Arrow pointing left (<).

```
  rarrow : Curses.chtype ;
```

Arrow pointing right (>).

```
  darrow : Curses.chtype ;
  uarrow : Curses.chtype ;
```

Arrow pointing up (^).

```
  board : Curses.chtype ;
  lantern : Curses.chtype ;
  block : Curses.chtype ;
```

Solid square block (#).

```
  s3 : Curses.chtype ;
```

Scan line 3 (-).

```
  s7 : Curses.chtype ;
```

Scan line 7 (-).

```
  lequal : Curses.chtype ;
```

Less-than-or-equal-to (<).

```
  gequal : Curses.chtype ;
```

Greater-or-equal-to (>).

```
  pi : Curses.chtype ;
```

Greek pi (*).

```
  nequal : Curses.chtype ;
```

Not-equal (!).

```
  sterling : Curses.chtype ;
```

Pound-Sterling symbol (f).

```
}
val bssb : acs -> Curses.chtype
val ssbb : acs -> Curses.chtype
val bbss : acs -> Curses.chtype
```

```
    val sbbs : acs -> Curses.chtype
    val sbss : acs -> Curses.chtype
    val sssb : acs -> Curses.chtype
    val ssbs : acs -> Curses.chtype
    val bsss : acs -> Curses.chtype
    val bsbs : acs -> Curses.chtype
    val sbsb : acs -> Curses.chtype
    val ssss : acs -> Curses.chtype
  end
```

Predefined characters.

```
val get_acs_codes : unit -> Acs.acs
```

Get the predefined characters.

1.3.1 Displaying characters

```
val addch : chtype -> err
```

Add a character at the current position, then advance the cursor.

```
val waddch : window -> chtype -> err
val mvaddch : int -> int -> chtype -> err
val mvwaddch : window -> int -> int -> chtype -> err
val echochar : chtype -> err
```

`echochar ch` is equivalent to `addch ch` followed by `refresh ()`.

```
val wechochar : window -> chtype -> err
val addchstr : chtype array -> err
```

Add a sequence of characters at the current position. See also `addstr`.

```
val waddchstr : window -> chtype array -> err
val mvaddchstr : int -> int -> chtype array -> err
val mvwaddchstr : window -> int -> int -> chtype array -> err
val addchnstr : chtype array -> int -> int -> err
val waddchnstr : window -> chtype array -> int -> int -> err
val mvaddchnstr : int -> int -> chtype array -> int -> int -> err
val mvwaddchnstr :
  window ->
  int -> int -> chtype array -> int -> int -> err
val addstr : string -> err
```

Add a string at the current position.

```
val waddstr : window -> string -> err
val mvaddstr : int -> int -> string -> err
val mvwaddstr : window -> int -> int -> string -> err
val addnstr : string -> int -> int -> err
val waddnstr : window -> string -> int -> int -> err
val mvaddnstr : int -> int -> string -> int -> int -> err
val mvwaddnstr : window -> int -> int -> string -> int -> int -> err
val insch : chtype -> err
```

Insert a character before cursor.

```
val winsch : window -> chtype -> err
val mvinsch : int -> int -> chtype -> err
val mvwinsch : window -> int -> int -> chtype -> err
val insstr : string -> err
```

Insert a string before cursor.

```
val winsstr : window -> string -> err
val mvinsstr : int -> int -> string -> err
val mvwinsstr : window -> int -> int -> string -> err
val insnstr : string -> int -> int -> err
val winsnstr : window -> string -> int -> int -> err
val mvinsnstr : int -> int -> string -> int -> int -> err
val mvwinsnstr : window -> int -> int -> string -> int -> int -> err
val delch : unit -> err
```

Delete a character.

```
val wdelch : window -> err
val mvdelch : int -> int -> err
val mvwdelch : window -> int -> int -> err
```

1.3.2 Attributes

```
module A :
  sig
    val normal : int
```

Normal display (no highlight).

```
    val attributes : int
    val chartext : int
```

Bit-mask to extract a character.

```
val color : int
val standout : int
```

Best highlighting mode of the terminal.

```
val underline : int
```

Underlining.

```
val reverse : int
```

Reverse video.

```
val blink : int
```

Blinking.

```
val dim : int
```

Half bright.

```
val bold : int
```

Extra bright or bold.

```
val altcharset : int
```

Alternate character set.

```
val invis : int
```

Invisible or blank mode.

```
val protect : int
```

Protected mode.

```
val horizontal : int
val left : int
val low : int
val right : int
val top : int
val vertical : int
val combine : int list -> int
val color_pair : int -> int
```

Color-pair number `n`.

```
val pair_number : int -> int
```

Get the pair number associated with the `color_pair n` attribute.

```
  end
```

Attributes.

```
module WA :
  sig
```

```
    val normal : int
```

Normal display (no highlight).

```
    val attributes : int
    val chartext : int
    val color : int
    val standout : int
```

Best highlighting mode of the terminal. Same as `attron A.standout`.

```
    val underline : int
```

Underlining.

```
    val reverse : int
```

Reverse video.

```
    val blink : int
```

Blinking.

```
    val dim : int
```

Half bright.

```
    val bold : int
```

Extra bright or bold.

```
    val altcharset : int
```

Alternate character set.

```
    val invis : int
    val protect : int
    val horizontal : int
    val left : int
    val low : int
    val right : int
    val top : int
```

```
    val vertical : int
    val combine : int list -> int
    val color_pair : int -> int
    val pair_number : int -> int
  end
```

New series of highlight attributes.

```
val attroff : int -> unit
```

Turn off the attributes given in argument (see the `A` module).

```
val wattroff : window -> int -> unit
val attron : int -> unit
```

Turn on the attributes given in argument.

```
val wattron : window -> int -> unit
val attrset : int -> unit
```

Set the attributes.

```
val wattrset : window -> int -> unit
val standend : unit -> unit
val wstandend : window -> unit
val standout : unit -> unit
val wstandout : window -> unit
val attr_off : attr_t -> unit
```

Turn off the attributes given in argument (see the `WA` module).

```
val wattr_off : window -> attr_t -> unit
val attr_on : attr_t -> unit
val wattr_on : window -> attr_t -> unit
val attr_set : attr_t -> int -> unit
val wattr_set : window -> attr_t -> int -> unit
val chgat : int -> attr_t -> int -> unit
```

`chgat n attr color` changes the attributes of `n` characters.

```
val wchgat : window -> int -> attr_t -> int -> unit
val mvchgat : int -> int -> int -> attr_t -> int -> unit
val mvwchgat : window -> int -> int -> int -> attr_t -> int -> unit
val inch : unit -> chtype
```

Get the attributes of the caracter at current position.

```
val winch : window -> chtype
val mvinch : int -> int -> chtype
val mvwinch : window -> int -> int -> chtype
val inchstr : chtype array -> err
```

Get the attributes of a sequence of characters.

```
val winchstr : window -> chtype array -> err
val mvinchstr : int -> int -> chtype array -> err
val mvwinchstr : window -> int -> int -> chtype array -> err
val inchnstr : chtype array -> int -> int -> err
val winchnstr : window -> chtype array -> int -> int -> err
val mvinchnstr : int -> int -> chtype array -> int -> int -> err
val mvwinchnstr : window ->
  int -> int -> chtype array -> int -> int -> err
val instr : string -> err
```

Get the attributes of a string.

```
val winstr : window -> string -> err
val mvinstr : int -> int -> string -> err
val mvwinstr : window -> int -> int -> string -> err
val innstr : string -> int -> int -> err
val winnstr : window -> string -> int -> int -> err
val mvinnstr : int -> int -> string -> int -> int -> err
val mvwinnstr : window -> int -> int -> string -> int -> int -> err
```

1.3.3 Background

```
val bkgdset : chtype -> unit
```

Set the background of the current character.

```
val wbkgdset : window -> chtype -> unit
val bkgd : chtype -> unit
```

Set the background of every character.

```
val wbkgd : window -> chtype -> unit
val getbkgd : window -> chtype
```

Get the current background.

1.3.4 Operations on lines

`val deleteln : unit -> err`

Delete a line.

`val wdeleteln : window -> err`

`val insdelln : int -> err`

`insdelln n` inserts `n` lines above the current line if `n` is positive or deletes `-n` lines if `n` is negative.

`val winsdelln : window -> int -> err`

`val insertln : unit -> err`

Insert a blank line above the current line.

`val winsertln : window -> err`

1.3.5 Characters input

```
module Key :
  sig
    val code_yes : int
    val min : int
    val break : int
    val down : int
    val up : int
    val left : int
    val right : int
    val home : int
    val backspace : int
    val f0 : int
    val dl : int
    val il : int
    val dc : int
    val ic : int
    val eic : int
    val clear : int
    val eos : int
    val eol : int
    val sf : int
    val sr : int
    val npage : int
```

```
val ppage : int
val stab : int
val ctab : int
val catab : int
val enter : int
val sreset : int
val reset : int
val print : int
val ll : int
val a1 : int
val a3 : int
val b2 : int
val c1 : int
val c3 : int
val btab : int
val beg : int
val cancel : int
val close : int
val command : int
val copy : int
val create : int
val end_ : int
val exit : int
val find : int
val help : int
val mark : int
val message : int
val move : int
val next : int
val open_ : int
val options : int
val previous : int
val redo : int
val reference : int
val refresh : int
val replace : int
val restart : int
val resume : int
```

```
    val save : int
    val sbeg : int
    val scancel : int
    val scommand : int
    val scopy : int
    val screate : int
    val sdc : int
    val sdl : int
    val select : int
    val send : int
    val seol : int
    val sexit : int
    val sfind : int
    val shelp : int
    val shome : int
    val sic : int
    val sleft : int
    val smessage : int
    val smove : int
    val snext : int
    val soptions : int
    val sprevious : int
    val sprint : int
    val sredo : int
    val sreplace : int
    val sright : int
    val srsume : int
    val ssave : int
    val ssuspend : int
    val sundo : int
    val suspend : int
    val undo : int
    val mouse : int
    val resize : int
    val max : int
    val f : int -> int
end
```

Special keys.

`val getch : unit -> int`

Read a character in a window.

`val wgetch : window -> int`

`val mvgetch : int -> int -> int`

`val mvwgetch : window -> int -> int -> int`

`val ungetch : int -> err`

`val getstr : string -> err`

Read a string in a window.

`val wgetstr : window -> string -> err`

`val mvgetstr : int -> int -> string -> err`

`val mvwgetstr : window -> int -> int -> string -> err`

`val getnstr : string -> int -> int -> err`

`val wgetnstr : window -> string -> int -> int -> err`

`val mvgetnstr : int -> int -> string -> int -> int -> err`

`val mvwgetnstr : window -> int -> int -> string -> int -> int -> err`

1.4 Windows

1.4.1 Window manipulations

`val newwin : int -> int -> int -> int -> window`

`newwin l c y x` create a new window with `l` lines, `c` columns. The upper left-hand corner is at (`x`,`y`).

`val delwin : window -> err`

Delete a window.

`val mvwin : window -> int -> int -> err`

Move a window.

`val subwin : window -> int -> int -> int -> int -> window`

`subwin l c y x` create a subwindow with `l` lines and `c` columns at screen-relative position (`x`,`y`).

`val derwin : window -> int -> int -> int -> int -> window`

Same as `subwin` excepting that the position (`x`,`y`) is relative to the parent window.

`val mvderwin : window -> int -> int -> err`

Move a derived windw.

```
val dupwin : window -> window
```

Duplicate a window.

```
val wsyncup : window -> unit
val syncok : window -> bool -> err
```

If `syncok` is called with `true` as second argument, `wsyncup` is called automatically whenever there is a change in the window.

```
val wcursyncup : window -> unit
val wsyncdown : window -> unit
val winch_handler_on : unit -> unit
val winch_handler_off : unit -> unit
val get_size : unit -> int * int
val get_size_fd : Unix.file_descr -> int * int
val null_window : window
```

1.4.2 Refresh control

```
val refresh : unit -> err
```

Refresh windows.

```
val wrefresh : window -> err
val wnoutrefresh : window -> err
val doupdate : unit -> err
val redrawwin : window -> err
val wredrawln : window -> int -> int -> err
val wresize : window -> int -> int -> err
val resizeterm : int -> int -> err
val scroll : window -> err
val scrl : int -> err
val wscrl : window -> int -> err
val touchwin : window -> err
val touchline : window -> int -> int -> err
val untouchwin : window -> err
val wtouchln : window -> int -> int -> bool -> err
val is_linetouched : window -> int -> int
val is_wintouched : window -> bool
val erase : unit -> unit
```

Clear a window.

```
val werase : window -> unit
val clear : unit -> unit
val wclear : window -> unit
val clrtobot : unit -> unit
val wclrtobot : window -> unit
val clrtoeol : unit -> unit
val wclrtoeol : window -> unit
```

1.4.3 Overlapped windows

```
val overlay : window -> window -> err
```
`overlay srcwin dstwin` overlays `srcwin` on top of `dstwin`.

```
val overwrite : window -> window -> err
val copywin :
  window ->
  window -> int -> int -> int -> int -> int -> bool -> err
```

1.4.4 Decorations

```
val border :
  chtype ->
  chtype ->
  chtype ->
  chtype ->
  chtype -> chtype -> chtype -> chtype -> unit
```
Draw a box around the edges of a window.

```
val wborder :
  window ->
  chtype ->
  chtype ->
  chtype ->
  chtype ->
  chtype -> chtype -> chtype -> chtype -> unit
val box : window -> chtype -> chtype -> unit
```
Draw a box.

```
val hline : chtype -> int -> unit
```
Draw an horizontal line.

```
val whline : window -> chtype -> int -> unit
val mvhline : int -> int -> chtype -> int -> unit
val mvwhline : window -> int -> int -> chtype -> int -> unit
val vline : chtype -> int -> unit
```

Draw a vertical line.

```
val wvline : window -> chtype -> int -> unit
val mvvline : int -> int -> chtype -> int -> unit
val mvwvline : window -> int -> int -> chtype -> int -> unit
```

1.4.5 Pads

A pad is like a window except that it is not restricted by the screen size, and is not necessarily associated with a particular part of the screen.

```
val newpad : int -> int -> window
```

Create a new pad.

```
val subpad : window -> int -> int -> int -> int -> window
val prefresh : window -> int -> int -> int -> int -> int -> int -> err
val pnoutrefresh : window -> int -> int -> int -> int -> int -> int -> err
val pechochar : window -> chtype -> err
```

1.5 Colors

```
module Color :
  sig
     val black : int
     val red : int
     val green : int
     val yellow : int
     val blue : int
     val magenta : int
     val cyan : int
     val white : int
  end
```

Colors.

```
val start_color : unit -> err
val use_default_colors : unit -> err
val init_pair : int -> int -> int -> err
val init_color : int -> int -> int -> int -> err
val has_colors : unit -> bool
val can_change_color : unit -> bool
val color_content : int -> int * int * int
val pair_content : int -> int * int
val colors : unit -> int
val color_pairs : unit -> int
```

1.6 Input/output options

1.6.1 Input options

`val cbreak : unit -> err`

Disable line buffering.

`val halfdelay : int -> err`

Similar to `cbreak` but with delay.

`val nocbreak : unit -> err`

Enable line buffering (waits for characters until newline is typed).

`val echo : unit -> err`

Don't echo typed characters.

`val noecho : unit -> err`

Echo typed characters.

```
val intrflush : window -> bool -> err
val keypad : window -> bool -> err
val meta : window -> bool -> err
val nodelay : window -> bool -> err
val raw : unit -> err
val noraw : unit -> err
val noqiflush : unit -> unit
val qiflush : unit -> unit
val notimeout : window -> bool -> err
val timeout : int -> unit
val wtimeout : window -> int -> unit
val typeahead : Unix.file_descr -> err
val notypeahead : unit -> err
```

1.6.2 Output options

`val clearok : window -> bool -> unit`

If called with `true` as second argument, the next call to `wrefresh` with this window will clear the streen completely and redraw the entire screen from scratch.

```
val idlok : window -> bool -> unit
val idcok : window -> bool -> unit
val immedok : window -> bool -> unit
val leaveok : window -> bool -> unit
```

```
val setscrreg : int -> int -> err
val wsetscrreg : window -> int -> int -> err
val scrollok : window -> bool -> unit
val nl : unit -> unit
val nonl : unit -> unit
```

1.7 Soft-label keys

```
val slk_init : int -> err
```
Initialize soft labels.

```
val slk_set : int -> string -> int -> err
val slk_refresh : unit -> err
val slk_noutrefresh : unit -> err
val slk_label : int -> string
val slk_clear : unit -> err
val slk_restore : unit -> err
val slk_touch : unit -> err
val slk_attron : attr_t -> err
val slk_attroff : attr_t -> err
val slk_attrset : attr_t -> err
```

1.8 Mouse

```
val mousemask : int -> int * int
```
Sets the mouse mask.

1.9 Misc

```
val beep : unit -> err
```
Ring a bell.

```
val flash : unit -> err
```
Flash the screen.

```
val unctrl : chtype -> string
val keyname : int -> string
val filter : unit -> unit
val use_env : bool -> unit
val putwin : window -> Unix.file_descr -> err
val getwin : Unix.file_descr -> window
val delay_output : int -> err
val flushinp : unit -> unit
```

1.10 Screen manipulation

```
val scr_dump : string -> err
```
Dump the current screen to a file.

```
val scr_restore : string -> err
val scr_init : string -> err
val scr_set : string -> err
```

1.11 Terminal

```
val baudrate : unit -> int
```
Get the speed of a terminal (in bits per second).

```
val erasechar : unit -> char
```
Get user's current erase character.

```
val has_ic : unit -> bool
```
Has the terminal insert- and delete-character capabilites?

```
val has_il : unit -> bool
```
Has the terminal insert- and delete-line capabilites?

```
val killchar : unit -> char
```
Get user's current line kill character.

```
val longname : unit -> string
```
Get a description of the terminal.

```
val termattrs : unit -> attr_t
val termname : unit -> string
val tgetent : string -> bool
val tgetflag : string -> bool
val tgetnum : string -> int
val tgetstr : string -> bool
val tgoto : string -> int -> int -> string
val setupterm : string -> Unix.file_descr -> err
val setterm : string -> err
val cur_term : unit -> terminal
val set_curterm : terminal -> terminal
val del_curterm : terminal -> err
val restartterm : string -> Unix.file_descr -> err
```

```
val putp : string -> err
val vidattr : chtype -> err
val mvcur : int -> int -> int -> int -> err
val tigetflag : string -> bool
val tigetnum : string -> int
val tigetstr : string -> string
val tputs : string -> int -> (char -> unit) -> err
val vidputs : chtype -> (char -> unit) -> err
val tparm : string -> int array -> string
val bool_terminfo_variable : int -> string * string * string
val num_terminfo_variable : int -> string * string * string
val str_terminfo_variable : int -> string * string * string
val bool_terminfo_variables : (string, string * string) Hashtbl.t
val num_terminfo_variables : (string, string * string) Hashtbl.t
val str_terminfo_variables : (string, string * string) Hashtbl.t
```

1.12 Low-level curses routines

```
val def_prog_mode : unit -> unit
```

Save the current terminal modes as the "program" state for use by the `reser_prog_mod` and `reset_shell_mode` functions.

```
val def_shell_mode : unit -> unit
val reset_prog_mode : unit -> unit
val reset_shell_mode : unit -> unit
val resetty : unit -> unit
val savetty : unit -> unit
val getsyx : unit -> int * int
val setsyx : int -> int -> unit
val curs_set : int -> err
val napms : int -> unit
val ripoffline : bool -> unit
val get_ripoff : unit -> window * int
```

1.13 Configuration

```
module Curses_config :
  sig
    val wide_ncurses : bool
```

If `Curses` has been linked against a curses library with wide character support, then `wide_ncurses` is `true`.

`end`