Contents

PDCurses User's Guide 5

Curses Overview 5

Data Types and the <curses.h> Header 6

THE <curses.h> HEADER 6

DATA TYPES 6

VARIABLES 6

CONSTANTS 6

VIDEO ATTRIBUTES 7

For chtype: 7

For attr\_t: 7

VT100-compatible symbols -- box characters: 8

VT100-compatible symbols -- other: 8

Teletype 5410v1 symbols 8

Box character aliases: 8

COLORS 9

INPUT VALUES 9

PDCurses Definitions and Variables 19

PDCurses Text Attributes 20

PDCurses Functions 21

addch 21

Synopsis: 21

Description: 21

Return Value: 22

addchstr 23

Synopsis: 23

addstr 25

Synopsis: 25

attr 26

Synopsis: 26

The attrset() 26

beep 28

Synopsis: 28

bkgd 29

Synopsis: 29

Description: 29

border 31

Synopsis: 31

clear 33

Synopsis: 33

color 34

Synopsis: 34

init\_pair() 34

debug 36

Synopsis: 36

delch 37

Synopsis: 37

deleteln 38

Synopsis: 38

getch 39

Synopsis: 39

getstr 41

Synopsis: 41

getyx 43

Synopsis: 43

inch 44

Synopsis: 44

inchstr 45

Synopsis: 45

initscr 46

Synopsis: 46

inopts 48

Synopsis: 48

insch 50

Synopsis: 50

insstr 51

Synopsis: 51

instr 52

Synopsis: 52

kernel 53

Synopsis: 53

keyname 55

Synopsis: 55

mouse 56

Synopsis: 56

Description: 56

request\_mouse\_pos() 56

map\_button() 56

wmouse\_position() 56

getmouse() 57

wmouse\_trafo() 57

mouse\_trafo() 57

move 59

Synopsis: 59

Description: 59

Return Value: 59

outopts 60

Synopsis: 60

Description: 60

Return Value: 60

overlay 62

Synopsis: 62

Description: 62

overlay() 62

Return Value: 62

pad 62

Synopsis: 62

Description: 63

newpad() 63

subpad() 63

prefresh() 63

panel 65

Synopsis: 65

Description: 65

del\_panel() 65

hide\_panel() 65

move\_panel() 65

new\_panel() 65

panel\_above() 65

panel\_below() 66

panel\_hidden() 66

show\_panel() 66

top\_panel() 66

update\_panels() 66

printw 68

Synopsis: 68

Description: 68

Return Value: 68

refresh 69

Synopsis: 69

Description: 69

Return Value: 69

scanw 70

Synopsis: 70

Description: 70

Return Value 70

scr\_dump 71

Synopsis: 71

Description: 71

scroll 72

Synopsis: 72

Description: 72

Return Value: 72

slk 73

Synopsis: 73

Description: 73

termattr 75

Synopsis: 75

Description: 75

termname() 75

termattrs() 75

terminfo 77

Synopsis: 77

Description: 77

mvcur() 77

touch 78

Synopsis: 78

Description: 78

touchwin() 78

util 79

Synopsis: 79

Description: 79

delay\_output() 79

getcchar() 79

window 81

Synopsis: 81

Description: 81

newwin() 81

delwin() 81

mvwin() 81

subwin() 81

derwin() 81

mvderwin() 81

dupwin() 82

wsyncup() 82

If wsyncok() 82

resize\_window() 82

Return Value: 82

Errors: 82

clipboard 84

Synopsis: 84

Description: 84

PDC\_getclipboard() 84

PDC\_setclipboard 84

PDC\_clearclipboard() 84

pdckbd 85

Synopsis: 85

Description: 85

pdcsetsc 86

Synopsis: 86

int PDC\_set\_blink(bool blinkon); 86

void PDC\_set\_title(const char \*title); 86

Description: 86

PDC\_set\_blink() 86

PDC\_set\_title() 86

sb 87

Synopsis: 87

int sb\_init(void) 87

int sb\_set\_horz(int total, int viewport, int cur) 87

int sb\_set\_vert(int total, int viewport, int cur) 87

int sb\_get\_horz(int \*total, int \*viewport, int \*cur) 87

int sb\_get\_vert(int \*total, int \*viewport, int \*cur) 87

int sb\_refresh(void); 87

Description: 87

X11 Considerations 88

Interaction with stdio 88

X Resources 88

Using Resources 91

Compose Keys for Latin-1 92

Deprecated 94

SDL Considerations 95

Fonts 95

Backgrounds 95

Icons 96

Screen size 96

Integration with SDL 96

Interaction with stdio 97

# PDCurses User's Guide

=====================

## Curses Overview

---------------

The X/Open Curses Interface Definition describes a set of C-Language

functions that provide screen-handling and updating, which are

collectively known as the curses library.

The curses library permits manipulation of data structures called

windows which may be thought of as two-dimensional arrays of

characters representing all or part of a terminal's screen. The

windows are manipulated using a procedural interface described

elsewhere. The curses package maintains a record of what characters

are on the screen. At the most basic level, manipulation is done with

the routines move() and addch() which are used to "move" the curses

around and add characters to the default window, stdscr, which

represents the whole screen.

An application may use these routines to add data to the window in any

convenient order. Once all data have been added, the routine

refresh() is called. The package then determines what changes have

been made which affect the screen. The screen contents are then

changed to reflect those characters now in the window, using a

sequence of operations optimized for the type of terminal in use.

At a higher level routines combining the actions of move() and addch()

are defined, as are routines to add whole strings and to perform

format conversions in the manner of printf().

Interfaces are also defined to erase the entire window and to specify

the attributes of individual characters in the window. Attributes

such as inverse video, underline and blink can be used on a

per-character basis.

New windows can be created by allowing the application to build

several images of the screen and display the appropriate one very

quickly. New windows are created using the routine newwin(). For

each routine that manipulates the default window, stdscr, there is a

corresponding routine prefixed with w to manipulate the contents of a

specified window; for example, move() and wmove(). In fact, move(...)

is functionally equivalent to wmove( stdscr, ...). This is similar to

the interface offered by printf(...) and fprintf(stdout, ...).

Windows do not have to correspond to the entire screen. It is

possible to create smaller windows, and also to indicate that the

window is only partially visible on the screen. Furthermore, large

windows or pads, which are bigger than the actual screen size, may be

created.

Interfaces are also defined to allow input character manipulation and

to disable and enable many input attributes: character echo, single

character input with or without signal processing (cbreak or raw

modes), carriage returns mapping to newlines, screen scrolling, etc.

## Data Types and the <curses.h> Header

------------------------------------

The data types supported by curses are described in this section.

As the library supports a procedural interface to the data types, actual

structure contents are not described. All curses data are manipulated

using the routines provided.

### THE <curses.h> HEADER

The <curses.h> header defines various constants and declares the data

types that are available to the application.

### DATA TYPES

The following data types are declared:

WINDOW \* pointer to screen representation

SCREEN \* pointer to terminal descriptor

bool boolean data type

chtype representation of a character in a window

cchar\_t the wide-character equivalent of chtype

attr\_t for WA\_-style attributes

The actual WINDOW and SCREEN objects used to store information are

created by the corresponding routines and a pointer to them is provided.

All manipulation is through that pointer.

### VARIABLES

The following variables are defined:

LINES number of lines on terminal screen

COLS number of columns on terminal screen

stdscr pointer to the default screen window

curscr pointer to the current screen image

SP pointer to the current SCREEN struct

Mouse\_status status of the mouse

COLORS number of colors available

COLOR\_PAIRS number of color pairs available

TABSIZE size of one TAB block

acs\_map[] alternate character set map

ttytype[] terminal name/description

### CONSTANTS

The following constants are defined:

GENERAL

FALSE boolean false value

TRUE boolean true value

NULL zero pointer value

ERR value returned on error condition

OK value returned on successful completion

### VIDEO ATTRIBUTES

Normally, attributes are a property of the character.

#### For chtype:

A\_ALTCHARSET use the alternate character set

A\_BLINK bright background or blinking

A\_BOLD bright foreground or bold

A\_DIM half bright -- no effect in PDCurses

A\_INVIS invisible

A\_ITALIC italic

A\_LEFTLINE line along the left edge

A\_PROTECT protected (?) -- PDCurses renders this as a

combination of the \*LINE attributes

A\_REVERSE reverse video

A\_RIGHTLINE line along the right edge

A\_STANDOUT terminal's best highlighting mode

A\_UNDERLINE underline

A\_ATTRIBUTES bit-mask to extract attributes

A\_CHARTEXT bit-mask to extract a character

A\_COLOR bit-mask to extract a color-pair

Not all attributes will work on all terminals. A\_RIGHTLINE, A\_LEFTLINE

and A\_ITALIC are specific to PDCurses. A\_INVIS and A\_ITALIC are given

the same value in PDCurses.

#### For attr\_t:

WA\_ALTCHARSET same as A\_ALTCHARSET

WA\_BLINK same as A\_BLINK

WA\_BOLD same as A\_BOLD

WA\_DIM same as A\_DIM

WA\_INVIS same as A\_INVIS

WA\_LEFT same as A\_LEFTLINE

WA\_PROTECT same as A\_PROTECT

WA\_REVERSE same as A\_REVERSE

WA\_RIGHT same as A\_RIGHTLINE

WA\_STANDOUT same as A\_STANDOUT

WA\_UNDERLINE same as A\_UNDERLINE

Note that while A\_LEFTLINE and A\_RIGHTLINE are PDCurses-specific,

WA\_LEFT and WA\_RIGHT are standard. The following are also defined, for

compatibility, but currently have no effect in PDCurses: WA\_HORIZONTAL,

WA\_LOW, WA\_TOP, WA\_VERTICAL.

THE ALTERNATE CHARACTER SET

For use in chtypes and with related functions. These are a portable way

to represent graphics characters on different terminals.

#### VT100-compatible symbols -- box characters:

ACS\_ULCORNER upper left box corner

ACS\_LLCORNER lower left box corner

ACS\_URCORNER upper right box corner

ACS\_LRCORNER lower right box corner

ACS\_RTEE right "T"

ACS\_LTEE left "T"

ACS\_BTEE bottom "T"

ACS\_TTEE top "T"

ACS\_HLINE horizontal line

ACS\_VLINE vertical line

ACS\_PLUS plus sign, cross, or four-corner piece

#### VT100-compatible symbols -- other:

ACS\_S1 scan line 1

ACS\_S9 scan line 9

ACS\_DIAMOND diamond

ACS\_CKBOARD checkerboard -- 50% grey

ACS\_DEGREE degree symbol

ACS\_PLMINUS plus/minus sign

ACS\_BULLET bullet

Teletype 5410v1 symbols -- these are defined in SysV curses, but

are not well-supported by most terminals. Stick to VT100 characters

for optimum portability:

ACS\_LARROW left arrow

ACS\_RARROW right arrow

ACS\_DARROW down arrow

ACS\_UARROW up arrow

ACS\_BOARD checkerboard -- lighter (less dense) than

ACS\_CKBOARD

ACS\_LANTERN lantern symbol

ACS\_BLOCK solid block

That goes double for these -- undocumented SysV symbols. Don't use

them:

ACS\_S3 scan line 3

ACS\_S7 scan line 7

ACS\_LEQUAL less than or equal

ACS\_GEQUAL greater than or equal

ACS\_PI pi

ACS\_NEQUAL not equal

ACS\_STERLING pounds sterling symbol

#### Box character aliases:

ACS\_BSSB same as ACS\_ULCORNER

ACS\_SSBB same as ACS\_LLCORNER

ACS\_BBSS same as ACS\_URCORNER

ACS\_SBBS same as ACS\_LRCORNER

ACS\_SBSS same as ACS\_RTEE

ACS\_SSSB same as ACS\_LTEE

ACS\_SSBS same as ACS\_BTEE

ACS\_BSSS same as ACS\_TTEE

ACS\_BSBS same as ACS\_HLINE

ACS\_SBSB same as ACS\_VLINE

ACS\_SSSS same as ACS\_PLUS

For cchar\_t and wide-character functions, WACS\_ equivalents are also

defined.

### COLORS

For use with init\_pair(), color\_set(), etc.:

COLOR\_BLACK

COLOR\_BLUE

COLOR\_GREEN

COLOR\_CYAN

COLOR\_RED

COLOR\_MAGENTA

COLOR\_YELLOW

COLOR\_WHITE

Use these instead of numeric values. The definition of the colors

depends on the implementation of curses.

### INPUT VALUES

The following constants might be returned by getch() if keypad() has

been enabled. Note that not all of these may be supported on a

particular terminal:

KEY\_BREAK break key

KEY\_DOWN the four arrow keys

KEY\_UP

KEY\_LEFT

KEY\_RIGHT

KEY\_HOME home key (upward+left arrow)

KEY\_BACKSPACE backspace

KEY\_F0 function keys; space for 64 keys is reserved

KEY\_F(n) (KEY\_F0+(n))

KEY\_DL delete line

KEY\_IL insert line

KEY\_DC delete character

KEY\_IC insert character

KEY\_EIC exit insert character mode

KEY\_CLEAR clear screen

KEY\_EOS clear to end of screen

KEY\_EOL clear to end of line

KEY\_SF scroll 1 line forwards

KEY\_SR scroll 1 line backwards (reverse)

KEY\_NPAGE next page

KEY\_PPAGE previous page

KEY\_STAB set tab

KEY\_CTAB clear tab

KEY\_CATAB clear all tabs

KEY\_ENTER enter or send

KEY\_SRESET soft (partial) reset

KEY\_RESET reset or hard reset

KEY\_PRINT print or copy

KEY\_LL home down or bottom (lower left)

KEY\_A1 upper left of virtual keypad

KEY\_A3 upper right of virtual keypad

KEY\_B2 center of virtual keypad

KEY\_C1 lower left of virtual keypad

KEY\_C3 lower right of virtual keypad

KEY\_BTAB Back tab key

KEY\_BEG Beginning key

KEY\_CANCEL Cancel key

KEY\_CLOSE Close key

KEY\_COMMAND Cmd (command) key

KEY\_COPY Copy key

KEY\_CREATE Create key

KEY\_END End key

KEY\_EXIT Exit key

KEY\_FIND Find key

KEY\_HELP Help key

KEY\_MARK Mark key

KEY\_MESSAGE Message key

KEY\_MOVE Move key

KEY\_NEXT Next object key

KEY\_OPEN Open key

KEY\_OPTIONS Options key

KEY\_PREVIOUS Previous object key

KEY\_REDO Redo key

KEY\_REFERENCE Reference key

KEY\_REFRESH Refresh key

KEY\_REPLACE Replace key

KEY\_RESTART Restart key

KEY\_RESUME Resume key

KEY\_SAVE Save key

KEY\_SBEG Shifted beginning key

KEY\_SCANCEL Shifted cancel key

KEY\_SCOMMAND Shifted command key

KEY\_SCOPY Shifted copy key

KEY\_SCREATE Shifted create key

KEY\_SDC Shifted delete char key

KEY\_SDL Shifted delete line key

KEY\_SELECT Select key

KEY\_SEND Shifted end key

KEY\_SEOL Shifted clear line key

KEY\_SEXIT Shifted exit key

KEY\_SFIND Shifted find key

KEY\_SHELP Shifted help key

KEY\_SHOME Shifted home key

KEY\_SIC Shifted input key

KEY\_SLEFT Shifted left arrow key

KEY\_SMESSAGE Shifted message key

KEY\_SMOVE Shifted move key

KEY\_SNEXT Shifted next key

KEY\_SOPTIONS Shifted options key

KEY\_SPREVIOUS Shifted prev key

KEY\_SPRINT Shifted print key

KEY\_SREDO Shifted redo key

KEY\_SREPLACE Shifted replace key

KEY\_SRIGHT Shifted right arrow

KEY\_SRSUME Shifted resume key

KEY\_SSAVE Shifted save key

KEY\_SSUSPEND Shifted suspend key

KEY\_SUNDO Shifted undo key

KEY\_SUSPEND Suspend key

KEY\_UNDO Undo key

The virtual keypad is arranged like this:

A1 up A3

left B2 right

C1 down C3

This list is incomplete -- see curses.h for the full list, and use the

testcurs demo to see what values are actually returned. The above are

just the keys required by X/Open. In particular, PDCurses defines many

CTL\_ and ALT\_ combinations; these are not portable.

FUNCTIONS

The following table lists each curses routine and the name of the manual

page on which it is described.

Functions from the X/Open curses standard -- complete, except for

getch() and ungetch(), which are implemented as macros for DOS

compatibility:

Curses Function Manual Page Name

addch addch

addchnstr addchstr

addchstr addchstr

addnstr addstr

addstr addstr

attroff attr

attron attr

attrset attr

attr\_get attr

attr\_off attr

attr\_on attr

attr\_set attr

baudrate termattr

beep beep

bkgd bkgd

bkgdset bkgd

border border

box border

can\_change\_color color

cbreak inopts

chgat attr

clearok outopts

clear clear

clrtobot clear

clrtoeol clear

color\_content color

color\_set attr

copywin overlay

curs\_set kernel

def\_prog\_mode kernel

def\_shell\_mode kernel

del\_curterm terminfo

delay\_output util

delch delch

deleteln deleteln

delscreen I nitscr

delwin window

derwin window

doupdate refresh

dupwin window

echochar addch

echo inopts

endwin initscr

erasechar termattr

erase clear

filter util

flash beep

flushinp getch

getbkgd bkgd

getnstr getstr

getstr getstr

getwin scr\_dump

halfdelay inopts

has\_colors color

has\_ic termattr

has\_il termattr

hline border

idcok outopts

idlok outopts

immedok outopts

inchnstr inchstr

inchstr inchstr

inch inch

init\_color color

init\_pair color

initscr initscr

innstr instr

insch insch

insdelln deleteln

insertln deleteln

insnstr innstr

insstr innstr

instr instr

intrflush inopts

isendwin initscr

is\_linetouched touch

is\_wintouched touch

keyname keyname

keypad inopts

killchar termattr

leaveok outopts

longname termattr

meta inopts

move move

mvaddch addch

mvaddchnstr addchstr

mvaddchstr addchstr

mvaddnstr addstr

mvaddstr addstr

mvchgat attr

mvcur terminfo

mvdelch delch

mvderwin window

mvgetch getch

mvgetnstr getstr

mvgetstr getstr

mvhline border

mvinch inch

mvinchnstr inchstr

mvinchstr inchstr

mvinnstr instr

mvinsch insch

mvinsnstr insstr

mvinsstr insstr

mvinstr instr

mvprintw printw

mvscanw scanw

mvvline border

mvwaddchnstr addchstr

mvwaddchstr addchstr

mvwaddch addch

mvwaddnstr addstr

mvwaddstr addstr

mvwchgat attr

mvwdelch delch

mvwgetch getch

mvwgetnstr getstr

mvwgetstr getstr

mvwhline border

mvwinchnstr inchstr

mvwinchstr inchstr

mvwinch inch

mvwinnstr instr

mvwinsch insch

mvwinsnstr insstr

mvwinsstr insstr

mvwinstr instr

mvwin window

mvwprintw printw

mvwscanw scanw

mvwvline border

napms kernel

newpad pad

newterm initscr

newwin window

nl inopts

nocbreak inopts

nodelay inopts

noecho inopts

nonl inopts

noqiflush inopts

noraw inopts

notimeout inopts

overlay overlay

overwrite overlay

pair\_content color

pechochar pad

pnoutrefresh pad

prefresh pad

printw printw

putp terminfo

putwin scr\_dump

qiflush inopts

raw inopts

redrawwin refresh

refresh refresh

reset\_prog\_mode kernel

reset\_shell\_mode kernel

resetty kernel

restartterm terminfo

ripoffline kernel

savetty kernel

scanw scanw

scr\_dump scr\_dump

scr\_init scr\_dump

scr\_restore scr\_dump

scr\_set scr\_dump

scrl scroll

scroll scroll

scrollok outopts

set\_term initscr

setscrreg outopts

setterm terminfo

setupterm terminfo

slk\_attroff slk

slk\_attr\_off slk

slk\_attron slk

slk\_attr\_on slk

slk\_attrset slk

slk\_attr\_set slk

slk\_clear slk

slk\_color slk

slk\_init slk

slk\_label slk

slk\_noutrefresh slk

slk\_refresh slk

slk\_restore slk

slk\_set slk

slk\_touch slk

standend attr

standout attr

start\_color color

subpad pad

subwin window

syncok window

termattrs termattrs

term\_attrs termattrs

termname termattrs

tgetent termcap

tgetflag termcap

tgetnum termcap

tgetstr termcap

tgoto termcap

tigetflag terminfo

tigetnum terminfo

tigetstr terminfo

timeout inopts

touchline touch

touchwin touch

tparm terminfo

tputs terminfo

typeahead inopts

untouchwin touch

use\_env util

vidattr terminfo

vid\_attr terminfo

vidputs terminfo

vid\_puts terminfo

vline border

vw\_printw printw

vwprintw printw

vw\_scanw scanw

vwscanw scanw

waddchnstr addchstr

waddchstr addchstr

waddch addch

waddnstr addstr

waddstr addstr

wattroff attr

wattron attr

wattrset attr

wattr\_get attr

wattr\_off attr

wattr\_on attr

wattr\_set attr

wbkgdset bkgd

wbkgd bkgd

wborder border

wchgat attr

wclear clear

wclrtobot clear

wclrtoeol clear

wcolor\_set attr

wcursyncup window

wdelch delch

wdeleteln deleteln

wechochar addch

werase clear

wgetch getch

wgetnstr getstr

wgetstr getstr

whline border

winchnstr inchstr

winchstr inchstr

winch inch

winnstr instr

winsch insch

winsdelln deleteln

winsertln deleteln

winsnstr insstr

winsstr insstr

winstr instr

wmove move

wnoutrefresh refresh

wprintw printw

wredrawln refresh

wrefresh refresh

wscanw scanw

wscrl scroll

wsetscrreg outopts

wstandend attr

wstandout attr

wsyncdown window

wsyncup window

wtimeout inopts

wtouchln touch

wvline border

Wide-character functions from the X/Open standard -- these are only

available when PDCurses is built with PDC\_WIDE defined, and the

prototypes are only available from curses.h when PDC\_WIDE is defined

before its inclusion in your app:

addnwstr addstr

addwstr addstr

add\_wch addch

add\_wchnstr addchstr

add\_wchstr addchstr

border\_set border

box\_set border

echo\_wchar addch

erasewchar termattr

getbkgrnd bkgd

getcchar util

getn\_wstr getstr

get\_wch getch

get\_wstr getstr

hline\_set border

innwstr instr

ins\_nwstr insstr

ins\_wch insch

ins\_wstr insstr

inwstr instr

in\_wch inch

in\_wchnstr inchstr

in\_wchstr inchstr

key\_name keyname

killwchar termattr

mvaddnwstr addstr

mvaddwstr addstr

mvadd\_wch addch

mvadd\_wchnstr addchstr

mvadd\_wchstr addchstr

mvgetn\_wstr getstr

mvget\_wch getch

mvget\_wstr getstr

mvhline\_set border

mvinnwstr instr

mvins\_nwstr insstr

mvins\_wch insch

mvins\_wstr insstr

mvinwstr instr

mvwaddnwstr addstr

mvwaddwstr addstr

mvwadd\_wch addch

mvwadd\_wchnstr addchstr

mvwadd\_wchstr addchstr

mvwgetn\_wstr getstr

mvwget\_wch getch

mvwget\_wstr getstr

mvwhline\_set border

mvwinnwstr instr

mvwins\_nwstr insstr

mvwins\_wch insch

mvwins\_wstr insstr

mvwin\_wch inch

mvwin\_wchnstr inchstr

mvwin\_wchstr inchstr

mvwinwstr instr

mvwvline\_set border

pecho\_wchar pad

setcchar util

slk\_wset slk

unget\_wch getch

vline\_set border

waddnwstr addstr

waddwstr addstr

wadd\_wch addch

wadd\_wchnstr addchstr

wadd\_wchstr addchstr

wbkgrnd bkgd

wbkgrndset bkgd

wborder\_set border

wecho\_wchar addch

wgetbkgrnd bkgd

wgetn\_wstr getstr

wget\_wch getch

wget\_wstr getstr

whline\_set border

winnwstr instr

wins\_nwstr insstr

wins\_wch insch

wins\_wstr insstr

winwstr instr

win\_wch inch

win\_wchnstr inchstr

win\_wchstr inchstr

wunctrl util

wvline\_set border

Quasi-standard functions, from Sys V or BSD curses:

getattrs attr

getbegx getyx

getbegy getyx

getmaxx getyx

getmaxy getyx

getparx getyx

getparx getyx

traceoff debug

traceon debug

unctrl util

Classic PDCurses mouse functions, based on Sys V:

mouse\_set mouse

mouse\_on mouse

mouse\_off mouse

request\_mouse\_pos mouse

map\_button mouse

wmouse\_position mouse

getmouse mouse

getbmap mouse

Functions from ncurses:

assume\_default\_colors color

curses\_version initscr

has\_key keyname

use\_default\_colors color

wresize window

mouseinterval mouse

mousemask mouse

mouse\_trafo mouse

nc\_getmouse mouse

ungetmouse mouse

wenclose mouse

wmouse\_trafo mouse

PDCurses-specific functions -- avoid these in code that's intended to be

portable:

addrawch addch

insrawch insch

is\_termresized initscr

mvaddrawch addch

mvdeleteln deleteln

mvinsertln deleteln

mvinsrawch insch

mvwaddrawch addch

mvwdeleteln deleteln

mvwinsertln deleteln

mvwinsrawch insch

raw\_output outopts

resize\_term initscr

resize\_window window

slk\_wlabel slk

waddrawch addch

winsrawch insch

wordchar termattr

PDC\_debug debug

PDC\_ungetch getch

PDC\_set\_blink pdcsetsc

PDC\_set\_line\_color color

PDC\_set\_title pdcsetsc

PDC\_clearclipboard pdcclip

PDC\_freeclipboard pdcclip

PDC\_getclipboard pdcclip

PDC\_setclipboard pdcclip

PDC\_get\_input\_fd pdckbd

PDC\_get\_key\_modifiers getch

PDC\_return\_key\_modifiers getch

PDC\_save\_key\_modifiers getch

Functions specific to the X11 port of PDCurses:

Xinitscr initscr

XCursesExit -

sb\_init sb

sb\_set\_horz sb

sb\_set\_vert sb

sb\_get\_horz sb

sb\_get\_vert sb

sb\_refresh sb

--------------------------------------------------------------------------

## PDCurses Definitions and Variables

==================================

PDCurses definitions list: (Only define those needed)

XCURSES True if compiling for X11.

PDCurses portable platform definitions list:

PDC\_BUILD Defines API build version.

PDCURSES Enables access to PDCurses-only routines.

XOPEN Always true.

SYSVcurses True if you are compiling for SYSV portability.

BSDcurses True if you are compiling for BSD portability.

--------------------------------------------------------------------------

## PDCurses Text Attributes

========================

Originally, PDCurses used a short (16 bits) for its chtype. To include

color, a number of things had to be sacrificed from the strict Unix and

System V support. The main problem was fitting all character attributes

and color into an unsigned char (all 8 bits!).

Today, PDCurses by default uses a long (32 bits) for its chtype, as in

System V. The short chtype is still available, by undefining CHTYPE\_LONG

and rebuilding the library.

The following is the structure of a win->\_attrs chtype:

short form:

-------------------------------------------------

|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|

-------------------------------------------------

color number | attrs | character eg 'a'

The available non-color attributes are bold, reverse and blink. Others

have no effect. The high order char is an index into an array of

physical colors (defined in color.c) -- 32 foreground/background color

pairs (5 bits) plus 3 bits for other attributes.

long form:

----------------------------------------------------------------------------

|31|30|29|28|27|26|25|24|23|22|21|20|19|18|17|16|15|14|13|12|..| 3| 2| 1| 0|

----------------------------------------------------------------------------

color number | modifiers | character eg 'a'

The available non-color attributes are bold, underline, invisible,

right-line, left-line, protect, reverse and blink. 256 color pairs (8

bits), 8 bits for other attributes, and 16 bits for character data.

--------------------------------------------------------------------------

## PDCurses Functions

==================

### addch

#### Synopsis:

int addch(const chtype ch);

int waddch(WINDOW \*win, const chtype ch);

int mvaddch(int y, int x, const chtype ch);

int mvwaddch(WINDOW \*win, int y, int x, const chtype ch);

int echochar(const chtype ch);

int wechochar(WINDOW \*win, const chtype ch);

int addrawch(chtype ch);

int waddrawch(WINDOW \*win, chtype ch);

int mvaddrawch(int y, int x, chtype ch);

int mvwaddrawch(WINDOW \*win, int y, int x, chtype ch);

int add\_wch(const cchar\_t \*wch);

int wadd\_wch(WINDOW \*win, const cchar\_t \*wch);

int mvadd\_wch(int y, int x, const cchar\_t \*wch);

int mvwadd\_wch(WINDOW \*win, int y, int x, const cchar\_t \*wch);

int echo\_wchar(const cchar\_t \*wch);

int wecho\_wchar(WINDOW \*win, const cchar\_t \*wch);

#### Description:

addch() inserts the character ch into the default window

(stdscr) at the current cursor position, and the window cursor

is advanced. The character is of the type chtype, containing

both data and attributes. add\_wch() is the wide-character

version, taking a pointer to a cchar\_t.

waddch() is like addch(), but also lets you specify the window.

(This is in fact the core output routine.) wadd\_wch() is the

wide version.

mvaddch() moves the cursor to the specified (y, x) position, and

inserts the character ch into stdscr. mvadd\_wch() is the

wide version.

mvwaddch() moves the cursor to the specified position and

inserts the character ch into the specified window. mvwadd\_wch()

is the wide version.

echochar() inserts the character ch into stdscr at the current

cursor position and calls refresh(). echo\_wchar() is the wide

version.

wechochar() inserts the character ch into the specified window

and calls wrefresh(). wecho\_wchar() is the wide version.

addrawch(), waddrawch(), mvaddrawch() and mvwaddrawch() are

PDCurses-specific wrappers for addch() etc. that disable the

translation of control characters.

All these routines are similar to putchar(). The following

applies to all:

If the cursor moves on to the right margin, an automatic newline

is performed. If scrollok is enabled, and a character is added

to the bottom right corner of the screen, the scrolling region

will be scrolled up one line. If scrolling is not allowed, ERR

will be returned.

If ch is a tab, newline, or backspace, the cursor will be moved

appropriately within the window. If ch is a newline, the

clrtoeol routine is called before the cursor is moved to the

beginning of the next line. If newline mapping is off, the

cursor will be moved to the next line, but the x coordinate will

be unchanged. If ch is a tab the cursor is moved to the next

tab position within the window. If ch is another control

character, it will be drawn in the ^X notation. Calling the

inch() routine after adding a control character returns the

representation of the control character, not the control

character.

Video attributes can be combined with a character by ORing them

into the parameter. Text, including attributes, can be copied

from one place to another by using inch() and addch().

Note that in PDCurses, for now, a cchar\_t and a chtype are the

same. The text field is 16 bits wide, and is treated as Unicode

(UCS-2) when PDCurses is built with wide-character support

(define PDC\_WIDE). So, in functions that take a chtype, like

addch(), both the wide and narrow versions will handle Unicode.

But for portability, you should use the wide functions.

#### Return Value:

All functions return OK on success and ERR on error.

Portability X/Open BSD SYS V

addch Y Y Y

waddch Y Y Y

mvaddch Y Y Y

mvwaddch Y Y Y

echochar Y - 3.0

wechochar Y - 3.0

addrawch - - -

waddrawch - - -

mvaddrawch - - -

mvwaddrawch - - -

add\_wch Y

wadd\_wch Y

mvadd\_wch Y

mvwadd\_wch Y

echo\_wchar Y

wecho\_wchar Y

--------------------------------------------------------------------------

### addchstr

#### Synopsis:

int addchstr(const chtype \*ch);

int addchnstr(const chtype \*ch, int n);

int waddchstr(WINDOW \*win, const chtype \*ch);

int waddchnstr(WINDOW \*win, const chtype \*ch, int n);

int mvaddchstr(int y, int x, const chtype \*ch);

int mvaddchnstr(int y, int x, const chtype \*ch, int n);

int mvwaddchstr(WINDOW \*, int y, int x, const chtype \*ch);

int mvwaddchnstr(WINDOW \*, int y, int x, const chtype \*ch, int n);

int add\_wchstr(const cchar\_t \*wch);

int add\_wchnstr(const cchar\_t \*wch, int n);

int wadd\_wchstr(WINDOW \*win, const cchar\_t \*wch);

int wadd\_wchnstr(WINDOW \*win, const cchar\_t \*wch, int n);

int mvadd\_wchstr(int y, int x, const cchar\_t \*wch);

int mvadd\_wchnstr(int y, int x, const cchar\_t \*wch, int n);

int mvwadd\_wchstr(WINDOW \*win, int y, int x, const cchar\_t \*wch);

int mvwadd\_wchnstr(WINDOW \*win, int y, int x, const cchar\_t \*wch,

int n);

Description:

These routines write a chtype or cchar\_t string directly into

the window structure, starting at the current or specified

position. The four routines with n as the last argument copy at

most n elements, but no more than will fit on the line. If n =

-1 then the whole string is copied, up to the maximum number

that will fit on the line.

The cursor position is not advanced. These routines do not check

for newline or other special characters, nor does any line

wrapping occur.

Return Value:

All functions return OK or ERR.

Portability X/Open BSD SYS V

addchstr Y - 4.0

waddchstr Y - 4.0

mvaddchstr Y - 4.0

mvwaddchstr Y - 4.0

addchnstr Y - 4.0

waddchnstr Y - 4.0

mvaddchnstr Y - 4.0

mvwaddchnstr Y - 4.0

add\_wchstr Y

wadd\_wchstr Y

mvadd\_wchstr Y

mvwadd\_wchstr Y

add\_wchnstr Y

wadd\_wchnstr Y

mvadd\_wchnstr Y

mvwadd\_wchnstr Y

--------------------------------------------------------------------------

### addstr

#### Synopsis:

int addstr(const char \*str);

int addnstr(const char \*str, int n);

int waddstr(WINDOW \*win, const char \*str);

int waddnstr(WINDOW \*win, const char \*str, int n);

int mvaddstr(int y, int x, const char \*str);

int mvaddnstr(int y, int x, const char \*str, int n);

int mvwaddstr(WINDOW \*win, int y, int x, const char \*str);

int mvwaddnstr(WINDOW \*win, int y, int x, const char \*str, int n);

int addwstr(const wchar\_t \*wstr);

int addnwstr(const wchar\_t \*wstr, int n);

int waddwstr(WINDOW \*win, const wchar\_t \*wstr);

int waddnwstr(WINDOW \*win, const wchar\_t \*wstr, int n);

int mvaddwstr(int y, int x, const wchar\_t \*wstr);

int mvaddnwstr(int y, int x, const wchar\_t \*wstr, int n);

int mvwaddwstr(WINDOW \*win, int y, int x, const wchar\_t \*wstr);

int mvwaddnwstr(WINDOW \*win, int y, int x, const wchar\_t \*wstr, int n);

Description:

These routines write all the characters of the null-terminated

string str or wstr on the given window. The functionality is

equivalent to calling waddch() once for each character in the

string. The routines with n as the last argument write at most

n characters; if n is negative, then the entire string will be

added.

Return Value:

All functions return OK or ERR.

Portability X/Open BSD SYS V

addstr Y Y Y

waddstr Y Y Y

mvaddstr Y Y Y

mvwaddstr Y Y Y

addnstr Y - 4.0

waddnstr Y - 4.0

mvaddnstr Y - 4.0

mvwaddnstr Y - 4.0

addwstr Y

waddwstr Y

mvaddwstr Y

mvwaddwstr Y

addnwstr Y

waddnwstr Y

mvaddnwstr Y

mvwaddnwstr Y

--------------------------------------------------------------------------

### attr

#### Synopsis:

int attroff(chtype attrs);

int wattroff(WINDOW \*win, chtype attrs);

int attron(chtype attrs);

int wattron(WINDOW \*win, chtype attrs);

int attrset(chtype attrs);

int wattrset(WINDOW \*win, chtype attrs);

int standend(void);

int wstandend(WINDOW \*win);

int standout(void);

int wstandout(WINDOW \*win);

int color\_set(short color\_pair, void \*opts);

int wcolor\_set(WINDOW \*win, short color\_pair, void \*opts);

int attr\_get(attr\_t \*attrs, short \*color\_pair, void \*opts);

int attr\_off(attr\_t attrs, void \*opts);

int attr\_on(attr\_t attrs, void \*opts);

int attr\_set(attr\_t attrs, short color\_pair, void \*opts);

int wattr\_get(WINDOW \*win, attr\_t \*attrs, short \*color\_pair,

void \*opts);

int wattr\_off(WINDOW \*win, attr\_t attrs, void \*opts);

int wattr\_on(WINDOW \*win, attr\_t attrs, void \*opts);

int wattr\_set(WINDOW \*win, attr\_t attrs, short color\_pair,

void \*opts);

int chgat(int n, attr\_t attr, short color, const void \*opts);

int mvchgat(int y, int x, int n, attr\_t attr, short color,

const void \*opts);

int mvwchgat(WINDOW \*win, int y, int x, int n, attr\_t attr,

short color, const void \*opts);

int wchgat(WINDOW \*win, int n, attr\_t attr, short color,

const void \*opts);

chtype getattrs(WINDOW \*win);

#### Description:

These functions manipulate the current attributes and/or colors

of the named window. These attributes can be any combination

of A\_STANDOUT, A\_REVERSE, A\_BOLD, A\_DIM, A\_BLINK, A\_UNDERLINE.

These constants are defined in <curses.h> and can be combined

with the bitwise-OR operator (|).

The current attributes of a window are applied to all characters

that are written into the window with waddch(). Attributes are

a property of the character, and move with the character

through any scrolling and insert/delete line/character operations.

To the extent possible on the particular terminal, they will be

displayed as the graphic rendition of characters put on the

screen.

attrset()

This function sets the current attributes of the given

window to attrs. The attroff() function turns off the named

attributes without turning on or off any other attributes. The

attron() function turns on the named attributes without affecting

any others. The color\_set() function sets the window color to

the value of color\_pair.

The standout() function is the same as attron(A\_STANDOUT).

The standend() function is the same as attrset(A\_NORMAL); that

is, it turns off all attributes.

Return Value:

All functions return OK on success and ERR on error.

Portability X/Open BSD SYS V

attroff Y Y Y

wattroff Y Y Y

attron Y Y Y

wattron Y Y Y

attrset Y Y Y

wattrset Y Y Y

standend Y Y Y

wstandend Y Y Y

standout Y Y Y

wstandout Y Y Y

color\_set Y

wcolor\_set Y

attr\_get Y

wattr\_get Y

attr\_on Y

wattr\_on Y

attr\_off Y

wattr\_off Y

attr\_set Y

wattr\_set Y

chgat Y

wchgat Y

mvchgat Y

mvwchgat Y

getattrs -

--------------------------------------------------------------------------

### beep

#### Synopsis:

int beep(void);

int flash(void);

Description:

These routines are used to signal the terminal user. The beep()

function will sound the audible bell on the terminal, if possible

and if not, will flash the screen (visible bell).

The flash() function will flash the screen.

Return Value:

These functions return OK.

Portability X/Open BSD SYS V

beep Y Y Y

flash Y Y Y

--------------------------------------------------------------------------

### bkgd

#### Synopsis:

int bkgd(chtype ch);

void bkgdset(chtype ch);

chtype getbkgd(WINDOW \*win);

int wbkgd(WINDOW \*win, chtype ch);

void wbkgdset(WINDOW \*win, chtype ch);

int bkgrnd(const cchar\_t \*wch);

void bkgrndset(const cchar\_t \*wch);

int getbkgrnd(cchar\_t \*wch);

int wbkgrnd(WINDOW \*win, const cchar\_t \*wch);

void wbkgrndset(WINDOW \*win, const cchar\_t \*wch);

int wgetbkgrnd(WINDOW \*win, cchar\_t \*wch);

#### Description:

The bkgdset() and wbkgdset() routines manipulate the backgound

of the named window. Background is a chtype consisting of any

combination of attributes and non-blank characters that are

written into the window with waddch(). Both the character and

attribute parts of the background are combined with the blank

characters. The background becomes a property of the character

and moves with the character through any scrolling and

insert/delete line/character operations. To the extent possible

on a particular terminal, the attribute part of the background

is displayed as the graphic rendition of the character put on

the screen.

The bkgd() and wbkgd() routines combine the new background with

every position in the window. Background is any combination of

attributes and a character. Only the attribute part is used to

set the background of non-blank characters, while both character

and attributes are used for blank positions. To the extent

possible on a particular terminal, the attribute part of the

background is displayed as the graphic rendition of the

character put on the screen.

The attributes that are defined with the attrset()/attron() set

of functions take precedence over the background attributes if

there is a conflict (e.g., different color pairs).

Return Value:

bkgd() and wbkgd() return OK, unless the window is NULL, in

which case they return ERR.

Portability X/Open BSD SYS V

bkgd Y - 4.0

bkgdset Y - 4.0

getbkgd Y

wbkgd Y - 4.0

wbkgdset Y - 4.0

bkgrnd Y

bkgrndset Y

getbkgrnd Y

wbkgrnd Y

wbkgrndset Y

wgetbkgrnd Y

--------------------------------------------------------------------------

### border

#### Synopsis:

int border(chtype ls, chtype rs, chtype ts, chtype bs, chtype tl,

chtype tr, chtype bl, chtype br);

int wborder(WINDOW \*win, chtype ls, chtype rs, chtype ts,

chtype bs, chtype tl, chtype tr, chtype bl, chtype br);

int box(WINDOW \*win, chtype verch, chtype horch);

int hline(chtype ch, int n);

int vline(chtype ch, int n);

int whline(WINDOW \*win, chtype ch, int n);

int wvline(WINDOW \*win, chtype ch, int n);

int mvhline(int y, int x, chtype ch, int n);

int mvvline(int y, int x, chtype ch, int n);

int mvwhline(WINDOW \*win, int y, int x, chtype ch, int n);

int mvwvline(WINDOW \*win, int y, int x, chtype ch, int n);

int border\_set(const cchar\_t \*ls, const cchar\_t \*rs,

const cchar\_t \*ts, const cchar\_t \*bs,

const cchar\_t \*tl, const cchar\_t \*tr,

const cchar\_t \*bl, const cchar\_t \*br);

int wborder\_set(WINDOW \*win, const cchar\_t \*ls, const cchar\_t \*rs,

const cchar\_t \*ts, const cchar\_t \*bs,

const cchar\_t \*tl, const cchar\_t \*tr,

const cchar\_t \*bl, const cchar\_t \*br);

int box\_set(WINDOW \*win, const cchar\_t \*verch, const cchar\_t \*horch);

int hline\_set(const cchar\_t \*wch, int n);

int vline\_set(const cchar\_t \*wch, int n);

int whline\_set(WINDOW \*win, const cchar\_t \*wch, int n);

int wvline\_set(WINDOW \*win, const cchar\_t \*wch, int n);

int mvhline\_set(int y, int x, const cchar\_t \*wch, int n);

int mvvline\_set(int y, int x, const cchar\_t \*wch, int n);

int mvwhline\_set(WINDOW \*win, int y, int x, const cchar\_t \*wch, int n);

int mvwvline\_set(WINDOW \*win, int y, int x, const cchar\_t \*wch, int n);

Description:

border(), wborder(), and box() draw a border around the edge of

the window. If any argument is zero, an appropriate default is

used:

ls left side of border ACS\_VLINE

rs right side of border ACS\_VLINE

ts top side of border ACS\_HLINE

bs bottom side of border ACS\_HLINE

tl top left corner of border ACS\_ULCORNER

tr top right corner of border ACS\_URCORNER

bl bottom left corner of border ACS\_LLCORNER

br bottom right corner of border ACS\_LRCORNER

hline() and whline() draw a horizontal line, using ch, starting

from the current cursor position. The cursor position does not

change. The line is at most n characters long, or as many as

will fit in the window.

vline() and wvline() draw a vertical line, using ch, starting

from the current cursor position. The cursor position does not

change. The line is at most n characters long, or as many as

will fit in the window.

Return Value:

These functions return OK on success and ERR on error.

Portability X/Open BSD SYS V

border Y - 4.0

wborder Y - 4.0

box Y Y Y

hline Y - 4.0

vline Y - 4.0

whline Y - 4.0

wvline Y - 4.0

mvhline Y

mvvline Y

mvwhline Y

mvwvline Y

border\_set Y

wborder\_set Y

box\_set Y

hline\_set Y

vline\_set Y

whline\_set Y

wvline\_set Y

mvhline\_set Y

mvvline\_set Y

mvwhline\_set Y

mvwvline\_set Y

--------------------------------------------------------------------------

### clear

#### Synopsis:

int clear(void);

int wclear(WINDOW \*win);

int erase(void);

int werase(WINDOW \*win);

int clrtobot(void);

int wclrtobot(WINDOW \*win);

int clrtoeol(void);

int wclrtoeol(WINDOW \*win);

Description:

erase() and werase() copy blanks to every position of the

window.

clear() and wclear() are similar to erase() and werase(), except

they also call clearok() to ensure that the the screen is

cleared on the next call to wrefresh() for that window.

clrtobot() and wclrtobot() clear the screen from the current

cursor position to the end of the window.

clrtoeol() and wclrtoeol() clear the screen from the current

cursor position to the end of the current line.

Return Value:

All functions return OK on success and ERR on error.

Portability X/Open BSD SYS V

clear Y Y Y

wclear Y Y Y

erase Y Y Y

werase Y Y Y

clrtobot Y Y Y

wclrtobot Y Y Y

clrtoeol Y Y Y

wclrtoeol Y Y Y

--------------------------------------------------------------------------

### color

#### Synopsis:

int start\_color(void);

int init\_pair(short pair, short fg, short bg);

int init\_color(short color, short red, short green, short blue);

bool has\_colors(void);

bool can\_change\_color(void);

int color\_content(short color, short \*red, short \*green, short \*blue);

int pair\_content(short pair, short \*fg, short \*bg);

int assume\_default\_colors(int f, int b);

int use\_default\_colors(void);

int PDC\_set\_line\_color(short color);

Description:

To use these routines, start\_color() must be called, usually

immediately after initscr(). Colors are always used in pairs

refered to as color-pairs. A color-pair consists of a foreground

color and a background color. A color-pair is initialized with

init\_pair(). After it has been initialized, COLOR\_PAIR(n) can be

used like any other video attribute.

start\_color() initializes eight basic colors (black, red, green,

yellow, blue, magenta, cyan, and white), and two global

variables; COLORS and COLOR\_PAIRS (respectively defining the

maximum number of colors and color-pairs the terminal is capable

of displaying).

init\_pair() changes the definitions of a color-pair. The routine

takes three arguments: the number of the color-pair to be

redefined, and the new values of the foreground and background

colors. The value of color-pair must be between 0 and

COLOR\_PAIRS - 1, inclusive. The values of foreground and

background must be between 0 and COLORS - 1, inclusive. If the

color pair was previously initialized, the screen is refreshed

and all occurrences of that color-pair are changed to the new

definition.

has\_colors() indicates if the terminal supports, and can

maniplulate color. It returns TRUE or FALSE.

can\_change\_color() indicates if the terminal has the capability

to change the definition of its colors.

pair\_content() is used to determine what the colors of a given

color-pair consist of.

assume\_default\_colors() and use\_default\_colors() emulate the

ncurses extensions of the same names. assume\_default\_colors(f,

b) is essentially the same as init\_pair(0, f, b) (which isn't

allowed); it redefines the default colors. use\_default\_colors()

allows the use of -1 as a foreground or background color with

init\_pair(), and calls assume\_default\_colors(-1, -1); -1

represents the foreground or background color that the terminal

had at startup. If the environment variable PDC\_ORIGINAL\_COLORS

is set at the time start\_color() is called, that's equivalent to

calling use\_default\_colors().

PDC\_set\_line\_color() is used to set the color, globally, for

the color of the lines drawn for the attributes: A\_UNDERLINE,

A\_OVERLINE, A\_LEFTLINE and A\_RIGHTLINE. A value of -1 (the

default) indicates that the current foreground color should be

used.

NOTE: COLOR\_PAIR() and PAIR\_NUMBER() are implemented as macros.

Return Value:

All functions return OK on success and ERR on error, except for

has\_colors() and can\_change\_colors(), which return TRUE or FALSE.

Portability X/Open BSD SYS V

start\_color Y - 3.2

init\_pair Y - 3.2

init\_color Y - 3.2

has\_colors Y - 3.2

can\_change\_color Y - 3.2

color\_content Y - 3.2

pair\_content Y - 3.2

assume\_default\_colors - - -

use\_default\_colors - - -

PDC\_set\_line\_color - - -

--------------------------------------------------------------------------

### debug

#### Synopsis:

void traceon(void);

void traceoff(void);

void PDC\_debug(const char \*, ...);

Description:

traceon() and traceoff() toggle the recording of debugging

information to the file "trace". Although not standard, similar

functions are in some other curses implementations.

PDC\_debug() is the function that writes to the file, based on

whether traceon() has been called. It's used from the PDC\_LOG()

macro.

Portability X/Open BSD SYS V

traceon - - -

traceoff - - -

PDC\_debug - - -

--------------------------------------------------------------------------

### delch

#### Synopsis:

int delch(void);

int wdelch(WINDOW \*win);

int mvdelch(int y, int x);

int mvwdelch(WINDOW \*win, int y, int x);

Description:

The character under the cursor in the window is deleted. All

characters to the right on the same line are moved to the left

one position and the last character on the line is filled with

a blank. The cursor position does not change (after moving to

y, x if coordinates are specified).

Return Value:

All functions return OK on success and ERR on error.

Portability X/Open BSD SYS V

delch Y Y Y

wdelch Y Y Y

mvdelch Y Y Y

mvwdelch Y Y Y

--------------------------------------------------------------------------

### deleteln

#### Synopsis:

int deleteln(void);

int wdeleteln(WINDOW \*win);

int insdelln(int n);

int winsdelln(WINDOW \*win, int n);

int insertln(void);

int winsertln(WINDOW \*win);

int mvdeleteln(int y, int x);

int mvwdeleteln(WINDOW \*win, int y, int x);

int mvinsertln(int y, int x);

int mvwinsertln(WINDOW \*win, int y, int x);

Description:

With the deleteln() and wdeleteln() functions, the line under

the cursor in the window is deleted. All lines below the

current line are moved up one line. The bottom line of the

window is cleared. The cursor position does not change.

With the insertln() and winsertn() functions, a blank line is

inserted above the current line and the bottom line is lost.

mvdeleteln(), mvwdeleteln(), mvinsertln() and mvwinsertln()

allow moving the cursor and inserting/deleting in one call.

Return Value:

All functions return OK on success and ERR on error.

Portability X/Open BSD SYS V

deleteln Y Y Y

wdeleteln Y Y Y

mvdeleteln - - -

mvwdeleteln - - -

insdelln Y - 4.0

winsdelln Y - 4.0

insertln Y Y Y

winsertln Y Y Y

mvinsertln - - -

mvwinsertln - - -

--------------------------------------------------------------------------

### getch

#### Synopsis:

int getch(void);

int wgetch(WINDOW \*win);

int mvgetch(int y, int x);

int mvwgetch(WINDOW \*win, int y, int x);

int ungetch(int ch);

int flushinp(void);

int get\_wch(wint\_t \*wch);

int wget\_wch(WINDOW \*win, wint\_t \*wch);

int mvget\_wch(int y, int x, wint\_t \*wch);

int mvwget\_wch(WINDOW \*win, int y, int x, wint\_t \*wch);

int unget\_wch(const wchar\_t wch);

unsigned long PDC\_get\_key\_modifiers(void);

int PDC\_save\_key\_modifiers(bool flag);

int PDC\_return\_key\_modifiers(bool flag);

Description:

With the getch(), wgetch(), mvgetch(), and mvwgetch() functions,

a character is read from the terminal associated with the window.

In nodelay mode, if there is no input waiting, the value ERR is

returned. In delay mode, the program will hang until the system

passes text through to the program. Depending on the setting of

cbreak(), this will be after one character or after the first

newline. Unless noecho() has been set, the character will also

be echoed into the designated window.

If keypad() is TRUE, and a function key is pressed, the token for

that function key will be returned instead of the raw characters.

Possible function keys are defined in <curses.h> with integers

beginning with 0401, whose names begin with KEY\_.

If nodelay(win, TRUE) has been called on the window and no input

is waiting, the value ERR is returned.

The ungetch() function places ch back onto the input queue to be

returned by the next call to wgetch().

The flushinp() routine throws away any type-ahead that has been

typed by the user and has not yet been read by the program.

PDC\_get\_key\_modifiers() returns the keyboard modifiers (shift,

control, alt, numlock) effective at the time of the last getch()

call, if PDC\_save\_key\_modifiers(TRUE) has been called before the

getch(). Use the macros PDC\_KEY\_MODIFIER\_\* to determine which

modifier(s) were set. PDC\_return\_key\_modifiers() tells getch()

to return modifier keys pressed alone as keystrokes (KEY\_ALT\_L,

etc.). These may not work on all platforms.

NOTE: getch() and ungetch() are implemented as macros, to avoid

conflict with many DOS compiler's runtime libraries.

Return Value:

These functions return ERR or the value of the character, meta

character or function key token.

Portability X/Open BSD SYS V

getch Y Y Y

wgetch Y Y Y

mvgetch Y Y Y

mvwgetch Y Y Y

ungetch Y Y Y

flushinp Y Y Y

get\_wch Y

wget\_wch Y

mvget\_wch Y

mvwget\_wch Y

unget\_wch Y

PDC\_get\_key\_modifiers - - -

--------------------------------------------------------------------------

### getstr

#### Synopsis:

int getstr(char \*str);

int wgetstr(WINDOW \*win, char \*str);

int mvgetstr(int y, int x, char \*str);

int mvwgetstr(WINDOW \*win, int y, int x, char \*str);

int getnstr(char \*str, int n);

int wgetnstr(WINDOW \*win, char \*str, int n);

int mvgetnstr(int y, int x, char \*str, int n);

int mvwgetnstr(WINDOW \*win, int y, int x, char \*str, int n);

int get\_wstr(wint\_t \*wstr);

int wget\_wstr(WINDOW \*win, wint\_t \*wstr);

int mvget\_wstr(int y, int x, wint\_t \*wstr);

int mvwget\_wstr(WINDOW \*win, int, int, wint\_t \*wstr);

int getn\_wstr(wint\_t \*wstr, int n);

int wgetn\_wstr(WINDOW \*win, wint\_t \*wstr, int n);

int mvgetn\_wstr(int y, int x, wint\_t \*wstr, int n);

int mvwgetn\_wstr(WINDOW \*win, int y, int x, wint\_t \*wstr, int n);

Description:

The effect of getstr() is as though a series of calls to getch()

were made, until a newline or carriage return is received. The

resulting value is placed in the area pointed to by \*str. The

erase and kill characters are interpreted, as well as any

special keys, such as function keys.

With wgetnstr(), a series of characters are read until a newline

or carriage return is received. The resulting value is placed

in the area pointed to by the character pointer str. The erase

and kill characters are interpreted. This differs from

wgetstr() in that the number of characters read is limited by a

passed argument.

WARNING: There is no way to know how long the buffer passed to

wgetstr() is, so use wgetnstr() to avoid buffer overflows.

Return Value:

This functions return ERR on failure or any other value on

success.

Portability X/Open BSD SYS V

getstr Y Y Y

wgetstr Y Y Y

mvgetstr Y Y Y

mvwgetstr Y Y Y

getnstr Y - 4.0

wgetnstr Y - 4.0

mvgetnstr Y - -

mvwgetnstr Y - -

get\_wstr Y

wget\_wstr Y

mvget\_wstr Y

mvwget\_wstr Y

getn\_wstr Y

wgetn\_wstr Y

mvgetn\_wstr Y

mvwgetn\_wstr Y

--------------------------------------------------------------------------

### getyx

#### Synopsis:

void getyx(WINDOW \*win, int y, int x);

void getparyx(WINDOW \*win, int y, int x);

void getbegyx(WINDOW \*win, int y, int x);

void getmaxyx(WINDOW \*win, int y, int x);

int getbegy(WINDOW \*win);

int getbegx(WINDOW \*win);

int getcury(WINDOW \*win);

int getcurx(WINDOW \*win);

int getpary(WINDOW \*win);

int getparx(WINDOW \*win);

int getmaxy(WINDOW \*win);

int getmaxx(WINDOW \*win);

Description:

With the getyx() macro, the cursor position of the window is

placed in the two integer variables y and x. getbegyx() and

getmaxyx() return the current beginning coordinates and size of

the specified window respectively. getparyx() returns the

beginning coordinates of the parent's window if the specified

window is a sub-window otherwise -1 is returned. These functions

are implemented as macros.

The functions getbegy(), getbegx(), getcurx(), getcury(),

getmaxy(), getmaxx(), getpary(), and getparx() return the

appropriate coordinate or size values, or ERR in the case of a

NULL window.

Portability X/Open BSD SYS V

getyx Y Y Y

getparyx - - 4.0

getbegyx - - 3.0

getmaxyx - - 3.0

getbegy - - -

getbegx - - -

getcury - - -

getcurx - - -

getpary - - -

getparx - - -

getmaxy - - -

getmaxx - - -

--------------------------------------------------------------------------

### inch

#### Synopsis:

chtype inch(void);

chtype winch(WINDOW \*win);

chtype mvinch(int y, int x);

chtype mvwinch(WINDOW \*win, int y, int x);

int in\_wch(cchar\_t \*wcval);

int win\_wch(WINDOW \*win, cchar\_t \*wcval);

int mvin\_wch(int y, int x, cchar\_t \*wcval);

int mvwin\_wch(WINDOW \*win, int y, int x, cchar\_t \*wcval);

Description:

The inch() functions retrieve the character and attribute from

the current or specified window position, in the form of a

chtype. If a NULL window is specified, (chtype)ERR is returned.

The in\_wch() functions are the wide-character versions; instead

of returning a chtype, they store a cchar\_t at the address

specified by wcval, and return OK or ERR. (No value is stored

when ERR is returned.) Note that in PDCurses, chtype and cchar\_t

are the same.

Portability X/Open BSD SYS V

inch Y Y Y

winch Y Y Y

mvinch Y Y Y

mvwinch Y Y Y

in\_wch Y

win\_wch Y

mvin\_wch Y

mvwin\_wch Y

--------------------------------------------------------------------------

### inchstr

#### Synopsis:

int inchstr(chtype \*ch);

int inchnstr(chtype \*ch, int n);

int winchstr(WINDOW \*win, chtype \*ch);

int winchnstr(WINDOW \*win, chtype \*ch, int n);

int mvinchstr(int y, int x, chtype \*ch);

int mvinchnstr(int y, int x, chtype \*ch, int n);

int mvwinchstr(WINDOW \*, int y, int x, chtype \*ch);

int mvwinchnstr(WINDOW \*, int y, int x, chtype \*ch, int n);

int in\_wchstr(cchar\_t \*wch);

int in\_wchnstr(cchar\_t \*wch, int n);

int win\_wchstr(WINDOW \*win, cchar\_t \*wch);

int win\_wchnstr(WINDOW \*win, cchar\_t \*wch, int n);

int mvin\_wchstr(int y, int x, cchar\_t \*wch);

int mvin\_wchnstr(int y, int x, cchar\_t \*wch, int n);

int mvwin\_wchstr(WINDOW \*win, int y, int x, cchar\_t \*wch);

int mvwin\_wchnstr(WINDOW \*win, int y, int x, cchar\_t \*wch, int n);

Description:

These routines read a chtype or cchar\_t string from the window,

starting at the current or specified position, and ending at the

right margin, or after n elements, whichever is less.

Return Value:

All functions return the number of elements read, or ERR on

error.

Portability X/Open BSD SYS V

inchstr Y - 4.0

winchstr Y - 4.0

mvinchstr Y - 4.0

mvwinchstr Y - 4.0

inchnstr Y - 4.0

winchnstr Y - 4.0

mvinchnstr Y - 4.0

mvwinchnstr Y - 4.0

in\_wchstr Y

win\_wchstr Y

mvin\_wchstr Y

mvwin\_wchstr Y

in\_wchnstr Y

win\_wchnstr Y

mvin\_wchnstr Y

mvwin\_wchnstr Y

--------------------------------------------------------------------------

### initscr

#### Synopsis:

WINDOW \*initscr(void);

WINDOW \*Xinitscr(int argc, char \*argv[]);

int endwin(void);

bool isendwin(void);

SCREEN \*newterm(const char \*type, FILE \*outfd, FILE \*infd);

SCREEN \*set\_term(SCREEN \*new);

void delscreen(SCREEN \*sp);

int resize\_term(int nlines, int ncols);

bool is\_termresized(void);

const char \*curses\_version(void);

Description:

The first curses routine called should be initscr(). This will

determine the terminal type and initialize all curses data

structures. The initscr() function also arranges that the first

call to refresh() will clear the screen. If errors occur,

initscr() will write an appropriate error message to standard

error and exit.

A program should always call endwin() before exiting or

escaping from curses mode temporarily. This routine will

restore tty modes, move the cursor to the lower left corner

of the screen and reset the terminal into the proper non-visual

mode. To resume curses after a temporary escape, call refresh()

or doupdate().

The isendwin() function returns TRUE if endwin() has been called

without any subsequent calls to wrefresh(), and FALSE otherwise.

In some implementations of curses, newterm() allows the use of

multiple terminals. Here, it's just an alternative interface for

initscr(). It always returns SP, or NULL.

set\_term() does nothing meaningful in PDCurses, but is included

for compatibility with other curses implementations.

resize\_term() is effectively two functions: When called with

nonzero values for nlines and ncols, it attempts to resize the

screen to the given size. When called with (0, 0), it merely

adjusts the internal structures to match the current size after

the screen is resized by the user. On the currently supported

platforms, this functionality is mutually exclusive: X11 allows

user resizing, while DOS, OS/2 and Win32 allow programmatic

resizing. If you want to support user resizing, you should check

for getch() returning KEY\_RESIZE, and/or call is\_termresized()

at appropriate times; if either condition occurs, call

resize\_term(0, 0). Then, with either user or programmatic

resizing, you'll have to resize any windows you've created, as

appropriate; resize\_term() only handles stdscr and curscr.

is\_termresized() returns TRUE if the curses screen has been

resized by the user, and a call to resize\_term() is needed.

Checking for KEY\_RESIZE is generally preferable, unless you're

not handling the keyboard.

curses\_version() returns a string describing the version of

PDCurses.

Return Value:

All functions return NULL on error, except endwin(), which

returns ERR on error.

Portability X/Open BSD SYS V

initscr Y Y Y

endwin Y Y Y

isendwin Y - 3.0

newterm Y - Y

set\_term Y - Y

delscreen Y - 4.0

resize\_term - - -

is\_termresized - - -

curses\_version - - -

--------------------------------------------------------------------------

### inopts

#### Synopsis:

int cbreak(void);

int nocbreak(void);

int echo(void);

int noecho(void);

int halfdelay(int tenths);

int intrflush(WINDOW \*win, bool bf);

int keypad(WINDOW \*win, bool bf);

int meta(WINDOW \*win, bool bf);

int nl(void);

int nonl(void);

int nodelay(WINDOW \*win, bool bf);

int notimeout(WINDOW \*win, bool bf);

int raw(void);

int noraw(void);

void noqiflush(void);

void qiflush(void);

void timeout(int delay);

void wtimeout(WINDOW \*win, int delay);

int typeahead(int fildes);

int crmode(void);

int nocrmode(void);

Description:

cbreak() and nocbreak() put the terminal into and out of cbreak

mode. In cbreak mode, characters typed by the user are

immediately available to the program and erase/kill character

processing is not performed. When out of cbreak mode, the

terminal driver will buffer characters typed until a newline or

carriage return is typed. Interrupt and flow control characters

are unaffected by this mode. Initially the terminal may or may

not need be in cbreak mode.

echo() and noecho() control whether typed characters are echoed

by the input routine. Initially, input characters are echoed.

Subsequent calls to echo() and noecho() do not flush type-ahead.

halfdelay() is similar to cbreak(), but allows for a time limit

to be specified, in tenths of a second. This causes getch() to

block for that period before returning ERR if no key has been

received. tenths must be between 1 and 255.

The keypad() function changes the keypad option of the user's

terminal. If enabled (bf is TRUE), the user can press a function

key (such as the left arrow key) and getch() will return a

single value that represents the KEY\_LEFT function key.

If disabled, nothing will be returned.

The nodelay() function controls whether wgetch() is a

non-blocking call. If the option is enabled, and no input is

ready, wgetch() will return ERR. If disabled, wgetch() will hang

until input is ready.

The nl() function enables the translation of a carriage return

into a newline on input. The nonl() function disables it.

Initially, the translation does occur.

With raw() and noraw(), the terminal in placed into or out of

raw mode. Raw mode is similar to cbreak mode, in that

characters typed are immediately passed through to the user

program. The differences are that in raw mode, the INTR, QUIT,

SUSP, and STOP characters are passed through without being

interpreted, and without generating a signal. The behaviour of

the BREAK key depends on other parameters of the terminal drive

that are not set by curses.

In PDCurses, the meta() function sets raw mode on or off.

The timeout() and wtimeout() functions set blocking or

non-blocking reads for the specified window. The delay is

measured in milliseconds. If it's negative, a blocking read is

used; if zero, then non-blocking reads are done -- if no input

is waiting, ERR is returned immediately. If the delay is

positive, the read blocks for the delay period; if the period

expires, ERR is returned.

intrflush(), notimeout(), noqiflush(), qiflush() and typeahead()

do nothing in PDCurses, but are included for compatibility with

other curses implementations.

crmode() and nocrmode() are archaic equivalents to cbreak() and

nocbreak(), respectively.

Return Value:

All functions return OK on success and ERR on error.

Portability X/Open BSD SYS V

cbreak Y Y Y

nocbreak Y Y Y

echo Y Y Y

noecho Y Y Y

halfdelay Y - Y

intrflush Y - Y

keypad Y - Y

meta Y - Y

nl Y Y Y

nonl Y Y Y

nodelay Y - Y

notimeout Y - Y

raw Y Y Y

noraw Y Y Y

noqiflush Y - Y

qiflush Y - Y

timeout Y - Y

wtimeout Y - Y

typeahead Y - Y

crmode -

nocrmode -

--------------------------------------------------------------------------

### insch

#### Synopsis:

int insch(chtype ch);

int winsch(WINDOW \*win, chtype ch);

int mvinsch(int y, int x, chtype ch);

int mvwinsch(WINDOW \*win, int y, int x, chtype ch);

int insrawch(chtype ch);

int winsrawch(WINDOW \*win, chtype ch);

int mvinsrawch(int y, int x, chtype ch);

int mvwinsrawch(WINDOW \*win, int y, int x, chtype ch);

int ins\_wch(const cchar\_t \*wch);

int wins\_wch(WINDOW \*win, const cchar\_t \*wch);

int mvins\_wch(int y, int x, const cchar\_t \*wch);

int mvwins\_wch(WINDOW \*win, int y, int x, const cchar\_t \*wch);

Description:

The insch() functions insert a chtype into the window at the

current or specified cursor position. The cursor is NOT

advanced. A newline is equivalent to clrtoeol(); tabs are

expanded; other control characters are converted as with

unctrl().

The ins\_wch() functions are the wide-character

equivalents, taking cchar\_t pointers rather than chtypes.

Video attributes can be combined with a character by ORing

them into the parameter. Text, including attributes, can be

copied from one place to another using inch() and insch().

insrawch() etc. are PDCurses-specific wrappers for insch() etc.

that disable the translation of control characters.

Return Value:

All functions return OK on success and ERR on error.

Portability X/Open BSD SYS V

insch Y Y Y

winsch Y Y Y

mvinsch Y Y Y

mvwinsch Y Y Y

insrawch - - -

winsrawch - - -

ins\_wch Y

wins\_wch Y

mvins\_wch Y

mvwins\_wch Y

--------------------------------------------------------------------------

### insstr

#### Synopsis:

int insstr(const char \*str);

int insnstr(const char \*str, int n);

int winsstr(WINDOW \*win, const char \*str);

int winsnstr(WINDOW \*win, const char \*str, int n);

int mvinsstr(int y, int x, const char \*str);

int mvinsnstr(int y, int x, const char \*str, int n);

int mvwinsstr(WINDOW \*win, int y, int x, const char \*str);

int mvwinsnstr(WINDOW \*win, int y, int x, const char \*str, int n);

int ins\_wstr(const wchar\_t \*wstr);

int ins\_nwstr(const wchar\_t \*wstr, int n);

int wins\_wstr(WINDOW \*win, const wchar\_t \*wstr);

int wins\_nwstr(WINDOW \*win, const wchar\_t \*wstr, int n);

int mvins\_wstr(int y, int x, const wchar\_t \*wstr);

int mvins\_nwstr(int y, int x, const wchar\_t \*wstr, int n);

int mvwins\_wstr(WINDOW \*win, int y, int x, const wchar\_t \*wstr);

int mvwins\_nwstr(WINDOW \*win, int y, int x, const wchar\_t \*wstr, int n);

Description:

With these routines, a character string (as many characters as

will fit on the line) is inserted before the character under

the cursor. All characters to the right of the cursor are moved

to the right, with the possibility of the rightmost characters

on the line being lost. The cursor position does not change

(after moving to y, x, if specified). The routines with n as

the last argument insert at most n characters; if n is

negative, then the entire string is inserted.

Return Value:

All functions return OK on success and ERR on error.

Portability X/Open BSD SYS V

insstr Y - 4.0

winsstr Y - 4.0

mvinsstr Y - 4.0

mvwinsstr Y - 4.0

insnstr Y - 4.0

winsnstr Y - 4.0

mvinsnstr Y - 4.0

mvwinsnstr Y - 4.0

ins\_wstr Y

wins\_wstr Y

mvins\_wstr Y

mvwins\_wstr Y

ins\_nwstr Y

wins\_nwstr Y

mvins\_nwstr Y

mvwins\_nwstr Y

--------------------------------------------------------------------------

### instr

#### Synopsis:

int instr(char \*str);

int innstr(char \*str, int n);

int winstr(WINDOW \*win, char \*str);

int winnstr(WINDOW \*win, char \*str, int n);

int mvinstr(int y, int x, char \*str);

int mvinnstr(int y, int x, char \*str, int n);

int mvwinstr(WINDOW \*win, int y, int x, char \*str);

int mvwinnstr(WINDOW \*win, int y, int x, char \*str, int n);

int inwstr(wchar\_t \*wstr);

int innwstr(wchar\_t \*wstr, int n);

int winwstr(WINDOW \*win, wchar\_t \*wstr);

int winnwstr(WINDOW \*win, wchar\_t \*wstr, int n);

int mvinwstr(int y, int x, wchar\_t \*wstr);

int mvinnwstr(int y, int x, wchar\_t \*wstr, int n);

int mvwinwstr(WINDOW \*win, int y, int x, wchar\_t \*wstr);

int mvwinnwstr(WINDOW \*win, int y, int x, wchar\_t \*wstr, int n);

Description:

These functions take characters (or wide characters) from the

current or specified position in the window, and return them as

a string in str (or wstr). Attributes are ignored. The functions

with n as the last argument return a string at most n characters

long.

Return Value:

Upon successful completion, innstr(), mvinnstr(), mvwinnstr()

and winnstr() return the number of characters actually read into

the string; instr(), mvinstr(), mvwinstr() and winstr() return

OK. Otherwise, all these functions return ERR.

Portability X/Open BSD SYS V

instr Y - 4.0

winstr Y - 4.0

mvinstr Y - 4.0

mvwinstr Y - 4.0

innstr Y - 4.0

winnstr Y - 4.0

mvinnstr Y - 4.0

mvwinnstr Y - 4.0

inwstr Y

winwstr Y

mvinwstr Y

mvwinwstr Y

innwstr Y

winnwstr Y

mvinnwstr Y

mvwinnwstr Y

--------------------------------------------------------------------------

### kernel

#### Synopsis:

int def\_prog\_mode(void);

int def\_shell\_mode(void);

int reset\_prog\_mode(void);

int reset\_shell\_mode(void);

int resetty(void);

int savetty(void);

void getsyx(int y, int x);

void setsyx(int y, int x);

int ripoffline(int line, int (\*init)(WINDOW \*, int));

int curs\_set(int visibility);

int napms(int ms);

int draino(int ms);

int resetterm(void);

int fixterm(void);

int saveterm(void);

Description:

def\_prog\_mode() and def\_shell\_mode() save the current terminal

modes as the "program" (in curses) or "shell" (not in curses)

state for use by the reset\_prog\_mode() and reset\_shell\_mode()

functions. This is done automatically by initscr().

reset\_prog\_mode() and reset\_shell\_mode() restore the terminal to

"program" (in curses) or "shell" (not in curses) state. These

are done automatically by endwin() and doupdate() after an

endwin(), so they would normally not be called before these

functions.

savetty() and resetty() save and restore the state of the

terminal modes. savetty() saves the current state in a buffer,

and resetty() restores the state to what it was at the last call

to savetty().

getsyx() obtains the coordinates of the virtual screen cursor.

If leaveok() is currently TRUE, then -1, -1 is returned. If

lines have been removed from the top of the screen with

ripoffline(), then getsyx() includes those lines, so y and x

should only be used by setyx(). setyx() sets the cursor position

of the virtual screen to the y,x coordinates. If y, x are -1,

-1, leaveok() is set TRUE. The getsyx() and setsyx() routines

are designed to be used by a library routine that manipulates

curses windows, but does not want to change the position of the

cursor.

Note that getsyx() and setsyx() are defined as macros only.

curs\_set() alters the appearance of the text cursor. A value of

0 for visibility makes the cursor disappear; a value of 1 makes

the cursor appear "normal" (usually an underline) and 2 makes

the cursor "highly visible" (usually a block).

ripoffline() allows the user to reduce the size of stdscr by 1

line. If the value of line is positive, the line is removed

from the top of the screen; negative from the bottom. Up to 5

lines can be ripped off stdscr by calling ripoffline()

consecutively. The function argument, init, is called from

within initscr() or newterm(), so ripoffline() must be called

before either of these functions. The init function is passed a

pointer to a 1 line WINDOW, and the width of the window. Calling

ripoffline() with a NULL initialise function pointer is not

advised.

The napms() function suspends the program for the specified

number of milliseconds. draino() is an archaic equivalent.

FYI: It is unclear whether savetty() and resetty() are meant to

duplicate reset\_prog\_mode() and reset\_shell\_mode(), or be a

backing store type of operation. At this time, they're

implemented similarly to the reset\_\*\_mode() routines.

resetterm(), fixterm() and saveterm() are archaic equivalents

for reset\_shell\_mode(), reset\_prog\_mode() and def\_prog\_mode(),

respectively.

Return Value:

All functions return OK on success and ERR on error, except

curs\_set(), which returns the previous visibility.

Portability X/Open BSD SYS V

def\_prog\_mode Y Y Y

def\_shell\_mode Y Y Y

reset\_prog\_mode Y Y Y

reset\_shell\_mode Y Y Y

resetty Y Y Y

savetty Y Y Y

getsyx - - 3.0

setsyx - - 3.0

ripoffline Y - 3.0

curs\_set Y - 3.0

napms Y Y Y

draino -

resetterm -

fixterm -

saveterm -

--------------------------------------------------------------------------

### keyname

#### Synopsis:

char \*keyname(int key);

char \*key\_name(wchar\_t c);

bool has\_key(int key);

Description:

keyname() returns a string corresponding to the argument key.

key may be any key returned by wgetch().

key\_name() is the wide-character version. It takes a wchar\_t

parameter, but still returns a char \*.

has\_key() returns TRUE for recognized keys, FALSE otherwise.

This function is an ncurses extension.

Portability X/Open BSD SYS V

keyname Y - 3.0

key\_name Y

has\_key - - -

--------------------------------------------------------------------------

### mouse

#### Synopsis:

int mouse\_set(unsigned long mbe);

int mouse\_on(unsigned long mbe);

int mouse\_off(unsigned long mbe);

int request\_mouse\_pos(void);

int map\_button(unsigned long button);

void wmouse\_position(WINDOW \*win, int \*y, int \*x);

unsigned long getmouse(void);

unsigned long getbmap(void);

int mouseinterval(int wait);

bool wenclose(const WINDOW \*win, int y, int x);

bool wmouse\_trafo(const WINDOW \*win, int \*y, int \*x, bool to\_screen);

bool mouse\_trafo(int \*y, int \*x, bool to\_screen);

mmask\_t mousemask(mmask\_t mask, mmask\_t \*oldmask);

int nc\_getmouse(MEVENT \*event);

int ungetmouse(MEVENT \*event);

#### Description:

As of PDCurses 3.0, there are two separate mouse interfaces: the

classic interface, which is based on the undocumented Sys V

mouse functions; and an ncurses-compatible interface. Both are

active at all times, and you can mix and match functions from

each, though it's not recommended. The ncurses interface is

essentially an emulation layer built on top of the classic

interface; it's here to allow easier porting of ncurses apps.

The classic interface: mouse\_set(), mouse\_on(), mouse\_off(),

request\_mouse\_pos(), map\_button(), wmouse\_position(),

getmouse(), and getbmap(). An application using this interface

would start by calling mouse\_set() or mouse\_on() with a non-zero

value, often ALL\_MOUSE\_EVENTS. Then it would check for a

KEY\_MOUSE return from getch(). If found, it would call

request\_mouse\_pos() to get the current mouse status.

mouse\_set(), mouse\_on() and mouse\_off() are analagous to

attrset(), attron() and attroff(). These functions set the

mouse button events to trap. The button masks used in these

functions are defined in curses.h and can be or'ed together.

They are the group of masks starting with BUTTON1\_RELEASED.

request\_mouse\_pos() requests curses to fill in the Mouse\_status

structure with the current state of the mouse.

map\_button() enables the specified mouse action to activate the

Soft Label Keys if the action occurs over the area of the screen

where the Soft Label Keys are displayed. The mouse actions are

defined in curses.h in the group that starts with BUTTON\_RELEASED.

wmouse\_position() determines if the current mouse position is

within the window passed as an argument. If the mouse is

outside the current window, -1 is returned in the y and x

arguments; otherwise the y and x coordinates of the mouse

(relative to the top left corner of the window) are returned in

y and x.

getmouse() returns the current status of the trapped mouse

buttons as set by mouse\_set() or mouse\_on().

getbmap() returns the current status of the button action used

to map a mouse action to the Soft Label Keys as set by the

map\_button() function.

The ncurses interface: mouseinterval(), wenclose(),

wmouse\_trafo(), mouse\_trafo(), mousemask(), nc\_getmouse(), and

ungetmouse(). A typical application using this interface would

start by calling mousemask() with a non-zero value, often

ALL\_MOUSE\_EVENTS. Then it would check for a KEY\_MOUSE return

from getch(). If found, it would call nc\_getmouse() to get the

current mouse status.

mouseinterval() sets the timeout for a mouse click. On all

current platforms, PDCurses receives mouse button press and

release events, but must synthesize click events. It does this

by checking whether a release event is queued up after a press

event. If it gets a press event, and there are no more events

waiting, it will wait for the timeout interval, then check again

for a release. A press followed by a release is reported as

BUTTON\_CLICKED; otherwise it's passed through as BUTTON\_PRESSED.

The default timeout is 150ms; valid values are 0 (no clicks

reported) through 1000ms. In x11, the timeout can also be set

via the clickPeriod resource. The return value from

mouseinterval() is the old timeout. To check the old value

without setting a new one, call it with a parameter of -1. Note

that although there's no classic equivalent for this function

(apart from the clickPeriod resource), the value set applies in

both interfaces.

wenclose() reports whether the given screen-relative y, x

coordinates fall within the given window.

wmouse\_trafo() converts between screen-relative and window-

relative coordinates. A to\_screen parameter of TRUE means to

convert from window to screen; otherwise the reverse. The

function returns FALSE if the coordinates aren't within the

window, or if any of the parameters are NULL. The coordinates

have been converted when the function returns TRUE.

mouse\_trafo() is the stdscr version of wmouse\_trafo().

mousemask() is nearly equivalent to mouse\_set(), but instead of

OK/ERR, it returns the value of the mask after setting it. (This

isn't necessarily the same value passed in, since the mask could

be altered on some platforms.) And if the second parameter is a

non-null pointer, mousemask() stores the previous mask value

there. Also, since the ncurses interface doesn't work with

PDCurses' BUTTON\_MOVED events, mousemask() filters them out.

nc\_getmouse() returns the current mouse status in an MEVENT

struct. This is equivalent to ncurses' getmouse(), renamed to

avoid conflict with PDCurses' getmouse(). But if you define

NCURSES\_MOUSE\_VERSION (preferably as 2) before including

curses.h, it defines getmouse() to nc\_getmouse(), along with a

few other redefintions needed for compatibility with ncurses

code. nc\_getmouse() calls request\_mouse\_pos(), which (not

getmouse()) is the classic equivalent.

ungetmouse() is the mouse equivalent of ungetch(). However,

PDCurses doesn't maintain a queue of mouse events; only one can

be pushed back, and it can overwrite or be overwritten by real

mouse events.

Portability X/Open BSD SYS V

mouse\_set - - 4.0

mouse\_on - - 4.0

mouse\_off - - 4.0

request\_mouse\_pos - - 4.0

map\_button - - 4.0

wmouse\_position - - 4.0

getmouse - - 4.0

getbmap - - 4.0

mouseinterval - - -

wenclose - - -

wmouse\_trafo - - -

mouse\_trafo - - -

mousemask - - -

nc\_getmouse - - -

ungetmouse - - -

--------------------------------------------------------------------------

### move

#### Synopsis:

int move(int y, int x);

int wmove(WINDOW \*win, int y, int x);

#### Description:

The cursor associated with the window is moved to the given

location. This does not move the physical cursor of the

terminal until refresh() is called. The position specified is

relative to the upper left corner of the window, which is (0,0).

#### Return Value:

All functions return OK on success and ERR on error.

Portability X/Open BSD SYS V

move Y Y Y

wmove Y Y Y

--------------------------------------------------------------------------

### outopts

#### Synopsis:

int clearok(WINDOW \*win, bool bf);

int idlok(WINDOW \*win, bool bf);

void idcok(WINDOW \*win, bool bf);

void immedok(WINDOW \*win, bool bf);

int leaveok(WINDOW \*win, bool bf);

int setscrreg(int top, int bot);

int wsetscrreg(WINDOW \*win, int top, int bot);

int scrollok(WINDOW \*win, bool bf);

int raw\_output(bool bf);

#### Description:

With clearok(), if bf is TRUE, the next call to wrefresh() with

this window will clear the screen completely and redraw the

entire screen.

The immedok() routine, called with a second argument of TRUE,

causes an automatic wrefrsh() to be called on the window every

time a change is made to that window, due to calls like;

waddch(), wclrtoeol(), etc... Not surprisingly, this causes a

severe performance overhead.

Normally, the hardware cursor is left at the location of the

window being refreshed. leaveok() allows the cursor to be

left wherever the update happens to leave it. It is useful

for applications where the cursor is not used, since it reduces

the need for cursor motions. If possible, the cursor is made

invisible when this option is enabled.

The setscrreg() and wsetscrreg() functions allow the user to set

a software scrolling region in a window. The parameters 'top'

and 'bot' are the line numbers of the top and bottom margin of

the scrolling region. (Line 0 is the top line of the window.)

If this option and scrollok() are enabled, an attempt to move

off the bottom margin will cause all lines in the scrolling

region to scroll up one line. Only the text of the window is

scrolled.

idlok() and idcok() do nothing in PDCurses, but are provided for

compatibility with other curses implementations.

raw\_output() enables the output of raw characters using the

'standard' \*add\* and \*ins\* curses functions (that is, it

disables translation of control characters).

#### Return Value:

All functions return OK on success and ERR on error.

Portability X/Open BSD SYS V

clearok Y Y Y

idlok Y Y Y

idcok Y - 4.0

immedok Y - 4.0

leaveok Y Y Y

setscrreg Y Y Y

wsetscrreg Y Y Y

scrollok Y Y Y

raw\_output - - -

--------------------------------------------------------------------------

### overlay

#### Synopsis:

int overlay(const WINDOW \*src\_w, WINDOW \*dst\_w)

int overwrite(const WINDOW \*src\_w, WINDOW \*dst\_w)

int copywin(const WINDOW \*src\_w, WINDOW \*dst\_w, int src\_tr,

int src\_tc, int dst\_tr, int dst\_tc, int dst\_br,

int dst\_bc, bool overlay)

#### Description:

overlay() and overwrite() overlay src\_w on top of dst\_w; that

is, all text in src\_w is copied into dst\_w. The windows src\_w

and dst\_w are not required to be the same size. Those characters

in the source window that intersect with characters in the

destination window are copied to the destination window, so that

the characters appear in the same physical position on the

screen. The difference between the two functions is that

overlay() is non-destructive (blanks are not copied) while

overwrite() is destructive (blanks are copied).

copywin() is similar to overwrite() and overlay(), but copywin()

does not require that the two windows overlap. The arguments

src\_tc and src\_tr specify the top left corner of the region to

be copied to the destination window. The arguments dst\_tc,

dst\_tr, dst\_br, dst\_bc specify the region within the destination

window to where the copy is made. The argument overlay, if TRUE,

indicates that the copy is done non-destructively (as in

overlay()). Blanks in the source window are not copied to the

destination window. When overlay is FALSE, (as in overwrite()),

the copy is destructive; blanks are copied to the destination

window.

#### Return Value:

All functions return OK on success and ERR on error.

Portability X/Open BSD SYS V

overlay Y Y Y

overwrite Y Y Y

copywin Y - 3.0

--------------------------------------------------------------------------

### pad

#### Synopsis:

WINDOW \*newpad(int nlines, int ncols);

WINDOW \*subpad(WINDOW \*orig, int nlines, int ncols,

int begy, int begx);

int prefresh(WINDOW \*win, int py, int px, int sy1, int sx1,

int sy2, int sx2);

int pnoutrefresh(WINDOW \*w, int py, int px, int sy1, int sx1,

int sy2, int sx2);

int pechochar(WINDOW \*pad, chtype ch);

int pecho\_wchar(WINDOW \*pad, const cchar\_t \*wch);

#### Description:

newpad() creates a new pad data structure. A pad is a special

case of a window, which is not restricted by the screen size,

and is not necessarily associated with a particular part of the

screen. A pad can be used when a large window is needed, and

only a part of the window will be on the screen at one tme.

Automatic refreshes of pads (e.g., from scrolling or echoing of

input) do not occur. It is not legal to call refresh() with a

pad as an argument; the routines prefresh() or pnoutrefresh()

should be called instead. Note that these routines require

additional parameters to specify the part of the pad to be

displayed and the location on the screen to be used for display.

subpad() creates a new sub-pad within a pad. The dimensions of

the sub-pad are nlines lines and ncols columns. The sub-pad is

at position (begy, begx) in the the parent pad. This position

is relative to the pad, and not to the screen as with subwin.

The sub-pad is made in the middle of the pad orig, so that

changes made to either pad will affect both. When using this

routine, it will often be necessary to call touchwin() before

calling prefresh().

prefresh() copies the specified pad to the physical terminal

screen. It takes account of what is already displayed on the

screen to optimize cursor movement. pnoutrefresh() copies the

named pad to the virtual screen. It then compares the virtual

screen with the physical screen and performs the actual update.

These routines are analogous to wrefresh() and wnoutrefresh(),

just with pads instead of windows. Additional parameters are

also needed to indicate what part of the pad and screen are

involved. The upper left corner of the part of the pad to be

displayed is specified by py and px. The coordinates sy1, sx1,

sy2, and sx2 specify the edges of the screen rectangle that will

contain the selected part of the pad.

The lower right corner of the pad rectangle to be displayed is

calculated from the screen co-ordinates. This ensures that the

screen rectangle and the pad rectangle are the same size. Both

rectangles must be entirely contained within their respective

structures.

pechochar() is functionally equivalent to addch() followed by

a call to prefresh().

Return Value:

All functions return OK on success and ERR on error.

Portability X/Open BSD SYS V

newpad Y - Y

subpad Y - Y

prefresh Y - Y

pnoutrefresh Y - Y

pechochar Y - 3.0

pecho\_wchar Y

--------------------------------------------------------------------------

### panel

#### Synopsis:

int bottom\_panel(PANEL \*pan);

int del\_panel(PANEL \*pan);

int hide\_panel(PANEL \*pan);

int move\_panel(PANEL \*pan, int starty, int startx);

PANEL \*new\_panel(WINDOW \*win);

PANEL \*panel\_above(const PANEL \*pan);

PANEL \*panel\_below(const PANEL \*pan);

int panel\_hidden(const PANEL \*pan);

const void \*panel\_userptr(const PANEL \*pan);

WINDOW \*panel\_window(const PANEL \*pan);

int replace\_panel(PANEL \*pan, WINDOW \*win);

int set\_panel\_userptr(PANEL \*pan, const void \*uptr);

int show\_panel(PANEL \*pan);

int top\_panel(PANEL \*pan);

void update\_panels(void);

#### Description:

The panel library is built using the curses library, and any

program using panels routines must call one of the curses

initialization routines such as initscr(). A program using these

routines must be linked with the panels and curses libraries.

The header <panel.h> includes the header <curses.h>.

The panels package gives the applications programmer a way to

have depth relationships between curses windows; a curses window

is associated with every panel. The panels routines allow curses

windows to overlap without making visible the overlapped

portions of underlying windows. The initial curses window,

stdscr, lies beneath all panels. The set of currently visible

panels is the 'deck' of panels.

The panels package allows the applications programmer to create

panels, fetch and set their associated windows, shuffle panels

in the deck, and manipulate panels in other ways.

bottom\_panel() places pan at the bottom of the deck. The size,

location and contents of the panel are unchanged.

del\_panel() deletes pan, but not its associated winwow.

hide\_panel() removes a panel from the deck and thus hides it

from view.

move\_panel() move() the curses window associated with pan, so

that its upper lefthand corner is at the supplied coordinates.

(Do not use mvwin() on the window.)

new\_panel() creates a new panel associated with win and returns

the panel pointer. The new panel is placed at the top of the

deck.

panel\_above() returns a pointer to the panel in the deck above

pan, or NULL if pan is the top panel. If the value of pan passed

is NULL, this function returns a pointer to the bottom panel in

the deck.

panel\_below() returns a pointer to the panel in the deck below

pan, or NULL if pan is the bottom panel. If the value of pan

passed is NULL, this function returns a pointer to the top panel

in the deck.

panel\_hidden() returns OK if pan is hidden and ERR if it is not.

panel\_userptr() - Each panel has a user pointer available for

maintaining relevant information. This function returns a

pointer to that information previously set up by

set\_panel\_userptr().

panel\_window() returns a pointer to the curses window associated

with the panel.

replace\_panel() replaces the current window of pan with win.

set\_panel\_userptr() - Each panel has a user pointer available

for maintaining relevant information. This function sets the

value of that information.

show\_panel() makes a previously hidden panel visible and places

it back in the deck on top.

top\_panel() places pan on the top of the deck. The size,

location and contents of the panel are unchanged.

update\_panels() refreshes the virtual screen to reflect the

depth relationships between the panels in the deck. The user

must use doupdate() to refresh the physical screen.

Return Value:

Each routine that returns a pointer to an object returns NULL if

an error occurs. Each panel routine that returns an integer,

returns OK if it executes successfully and ERR if it does not.

Portability X/Open BSD SYS V

bottom\_panel - - Y

del\_panel - - Y

hide\_panel - - Y

move\_panel - - Y

new\_panel - - Y

panel\_above - - Y

panel\_below - - Y

panel\_hidden - - Y

panel\_userptr - - Y

panel\_window - - Y

replace\_panel - - Y

set\_panel\_userptr - - Y

show\_panel - - Y

top\_panel - - Y

update\_panels - - Y

Credits:

Original Author - Warren Tucker N4HGF

{gatech,emory}!n4hgf!wht -or- wht@n4hgf.Mt-Park.GA.US

--------------------------------------------------------------------------

### printw

#### Synopsis:

int printw(const char \*fmt, ...);

int wprintw(WINDOW \*win, const char \*fmt, ...);

int mvprintw(int y, int x, const char \*fmt, ...);

int mvwprintw(WINDOW \*win, int y, int x, const char \*fmt,...);

int vwprintw(WINDOW \*win, const char \*fmt, va\_list varglist);

int vw\_printw(WINDOW \*win, const char \*fmt, va\_list varglist);

#### Description:

The printw() functions add a formatted string to the window at

the current or specified cursor position. The format strings are

the same as used in the standard C library's printf(). (printw()

can be used as a drop-in replacement for printf().)

#### Return Value:

All functions return the number of characters printed, or

ERR on error.

Portability X/Open BSD SYS V

printw Y Y Y

wprintw Y Y Y

mvprintw Y Y Y

mvwprintw Y Y Y

vwprintw Y - 4.0

vw\_printw Y

--------------------------------------------------------------------------

### refresh

#### Synopsis:

int refresh(void);

int wrefresh(WINDOW \*win);

int wnoutrefresh(WINDOW \*win);

int doupdate(void);

int redrawwin(WINDOW \*win);

int wredrawln(WINDOW \*win, int beg\_line, int num\_lines);

#### Description:

wrefresh() copies the named window to the physical terminal

screen, taking into account what is already there in order to

optimize cursor movement. refresh() does the same, using stdscr.

These routines must be called to get any output on the terminal,

as other routines only manipulate data structures. Unless

leaveok() has been enabled, the physical cursor of the terminal

is left at the location of the window's cursor.

wnoutrefresh() and doupdate() allow multiple updates with more

efficiency than wrefresh() alone. wrefresh() works by first

calling wnoutrefresh(), which copies the named window to the

virtual screen. It then calls doupdate(), which compares the

virtual screen to the physical screen and does the actual

update. A series of calls to wrefresh() will result in

alternating calls to wnoutrefresh() and doupdate(), causing

several bursts of output to the screen. By first calling

wnoutrefresh() for each window, it is then possible to call

doupdate() only once.

In PDCurses, redrawwin() is equivalent to touchwin(), and

wredrawln() is the same as touchline(). In some other curses

implementations, there's a subtle distinction, but it has no

meaning in PDCurses.

#### Return Value:

All functions return OK on success and ERR on error.

Portability X/Open BSD SYS V

refresh Y Y Y

wrefresh Y Y Y

wnoutrefresh Y Y Y

doupdate Y Y Y

redrawwin Y - 4.0

wredrawln Y - 4.0

--------------------------------------------------------------------------

### scanw

#### Synopsis:

int scanw(const char \*fmt, ...);

int wscanw(WINDOW \*win, const char \*fmt, ...);

int mvscanw(int y, int x, const char \*fmt, ...);

int mvwscanw(WINDOW \*win, int y, int x, const char \*fmt, ...);

int vwscanw(WINDOW \*win, const char \*fmt, va\_list varglist);

int vw\_scanw(WINDOW \*win, const char \*fmt, va\_list varglist);

#### Description:

These routines correspond to scanf(). scanw() reads input from

the default window; wscanw() from the specified window.

mvscanw() and mvwscanw() move the cursor to the specified

position before reading.

wgetstr() is called to get a string from the window, and the

resulting line is used as input for the scan. All character

interpretation is carried out according to the scanf() function

rules.

Return Value:

Upon successful completion, the scanw, mvscanw, mvwscanw and

wscanw functions return the number of items successfully

matched. On end-of-file, they return EOF. Otherwise they

return ERR.

Portability X/Open BSD SYS V

scanw Y Y Y

wscanw Y Y Y

mvscanw Y Y Y

mvwscanw Y Y Y

vwscanw Y - 4.0

vw\_scanw Y

--------------------------------------------------------------------------

### scr\_dump

#### Synopsis:

int putwin(WINDOW \*win, FILE \*filep);

WINDOW \*getwin(FILE \*filep);

int scr\_dump(const char \*filename);

int scr\_init(const char \*filename);

int scr\_restore(const char \*filename);

int scr\_set(const char \*filename);

#### Description:

getwin() reads window-related data previously stored in a file

by putwin(). It then creates and initialises a new window using

that data.

putwin() writes all data associated with a window into a file,

using an unspecified format. This information can be retrieved

later using getwin().

scr\_dump() writes the current contents of the virtual screen to

the file named by filename in an unspecified format.

scr\_restore() function sets the virtual screen to the contents

of the file named by filename, which must have been written

using scr\_dump(). The next refresh operation restores the screen

to the way it looked in the dump file.

In PDCurses, scr\_init() does nothing, and scr\_set() is a synonym

for scr\_restore(). Also, scr\_dump() and scr\_restore() save and

load from curscr. This differs from some other implementations,

where scr\_init() works with curscr, and scr\_restore() works with

newscr; but the effect should be the same. (PDCurses has no

newscr.)

Return Value:

On successful completion, getwin() returns a pointer to the

window it created. Otherwise, it returns a null pointer. Other

functions return OK or ERR.

Portability X/Open BSD SYS V

putwin Y

getwin Y

scr\_dump Y

scr\_init Y

scr\_restore Y

scr\_set Y

--------------------------------------------------------------------------

### scroll

#### Synopsis:

int scroll(WINDOW \*win);

int scrl(int n);

int wscrl(WINDOW \*win, int n);

#### Description:

scroll() causes the window to scroll up one line. This involves

moving the lines in the window data strcture.

With a positive n, scrl() and wscrl() scroll the window up n

lines (line i + n becomes i); otherwise they scroll the window

down n lines.

For these functions to work, scrolling must be enabled via

scrollok(). Note also that scrolling is not allowed if the

supplied window is a pad.

#### Return Value:

All functions return OK on success and ERR on error.

Portability X/Open BSD SYS V

scroll Y Y Y

scrl Y - 4.0

wscrl Y - 4.0

--------------------------------------------------------------------------

### slk

#### Synopsis:

int slk\_init(int fmt);

int slk\_set(int labnum, const char \*label, int justify);

int slk\_refresh(void);

int slk\_noutrefresh(void);

char \*slk\_label(int labnum);

int slk\_clear(void);

int slk\_restore(void);

int slk\_touch(void);

int slk\_attron(const chtype attrs);

int slk\_attr\_on(const attr\_t attrs, void \*opts);

int slk\_attrset(const chtype attrs);

int slk\_attr\_set(const attr\_t attrs, short color\_pair, void \*opts);

int slk\_attroff(const chtype attrs);

int slk\_attr\_off(const attr\_t attrs, void \*opts);

int slk\_color(short color\_pair);

int slk\_wset(int labnum, const wchar\_t \*label, int justify);

int PDC\_mouse\_in\_slk(int y, int x);

void PDC\_slk\_free(void);

void PDC\_slk\_initialize(void);

wchar\_t \*slk\_wlabel(int labnum)

#### Description:

These functions manipulate a window that contain Soft Label Keys

(SLK). To use the SLK functions, a call to slk\_init() must be

made BEFORE initscr() or newterm(). slk\_init() removes 1 or 2

lines from the useable screen, depending on the format selected.

The line(s) removed from the screen are used as a separate

window, in which SLKs are displayed.

slk\_init() requires a single parameter which describes the

format of the SLKs as follows:

0 3-2-3 format

1 4-4 format

2 4-4-4 format (ncurses extension)

3 4-4-4 format with index line (ncurses extension)

2 lines used

55 5-5 format (pdcurses format)

The functions slk\_refresh(), slk\_noutrefresh() and slk\_touch()

are analagous to refresh(), noutrefresh() and touch() functions.

Return Value:

All functions return OK on success and ERR on error.

Portability X/Open BSD SYS V

slk\_init Y - Y

slk\_set Y - Y

slk\_refresh Y - Y

slk\_noutrefresh Y - Y

slk\_label Y - Y

slk\_clear Y - Y

slk\_restore Y - Y

slk\_touch Y - Y

slk\_attron Y - Y

slk\_attrset Y - Y

slk\_attroff Y - Y

slk\_attr\_on Y

slk\_attr\_set Y

slk\_attr\_off Y

slk\_wset Y

PDC\_mouse\_in\_slk - - -

PDC\_slk\_free - - -

PDC\_slk\_initialize - - -

slk\_wlabel - - -

--------------------------------------------------------------------------

### termattr

#### Synopsis:

int baudrate(void);

char erasechar(void);

bool has\_ic(void);

bool has\_il(void);

char killchar(void);

char \*longname(void);

chtype termattrs(void);

attr\_t term\_attrs(void);

char \*termname(void);

int erasewchar(wchar\_t \*ch);

int killwchar(wchar\_t \*ch);

char wordchar(void);

#### Description:

baudrate() is supposed to return the output speed of the

terminal. In PDCurses, it simply returns INT\_MAX.

has\_ic and has\_il() return TRUE. These functions have meaning in

some other implementations of curses.

erasechar() and killchar() return ^H and ^U, respectively -- the

ERASE and KILL characters. In other curses implementations,

these may vary by terminal type. erasewchar() and killwchar()

are the wide-character versions; they take a pointer to a

location in which to store the character, and return OK or ERR.

longname() returns a pointer to a static area containing a

verbose description of the current terminal. The maximum length

of the string is 128 characters. It is defined only after the

call to initscr() or newterm().

termname() returns a pointer to a static area containing a

short description of the current terminal (14 characters).

termattrs() returns a logical OR of all video attributes

supported by the terminal.

wordchar() is a PDCurses extension of the concept behind the

functions erasechar() and killchar(), returning the "delete

word" character, ^W.

Portability X/Open BSD SYS V

baudrate Y Y Y

erasechar Y Y Y

has\_ic Y Y Y

has\_il Y Y Y

killchar Y Y Y

longname Y Y Y

termattrs Y Y Y

termname Y Y Y

erasewchar Y

killwchar Y

term\_attrs Y

wordchar - - -

--------------------------------------------------------------------------

### terminfo

#### Synopsis:

int mvcur(int oldrow, int oldcol, int newrow, int newcol);

int del\_curterm(TERMINAL \*);

int putp(const char \*);

int restartterm(const char \*, int, int \*);

TERMINAL \*set\_curterm(TERMINAL \*);

int setterm(const char \*term);

int setupterm(const char \*, int, int \*);

int tgetent(char \*, const char \*);

int tgetflag(const char \*);

int tgetnum(const char \*);

char \*tgetstr(const char \*, char \*\*);

char \*tgoto(const char \*, int, int);

int tigetflag(const char \*);

int tigetnum(const char \*);

char \*tigetstr(const char \*);

char \*tparm(const char \*,long, long, long, long, long, long,

long, long, long);

int tputs(const char \*, int, int (\*)(int));

int vidattr(chtype attr);

int vid\_attr(attr\_t attr, short color\_pair, void \*opt);

int vidputs(chtype attr, int (\*putfunc)(int));

int vid\_puts(attr\_t attr, short color\_pair, void \*opt,

int (\*putfunc)(int));

#### Description:

mvcur() lets you move the physical cursor without updating any

window cursor positions. It returns OK or ERR.

The rest of these functions are currently implemented as stubs,

returning the appropriate errors and doing nothing else.

Portability X/Open BSD SYS V

mvcur Y Y Y

--------------------------------------------------------------------------

### touch

#### Synopsis:

int touchwin(WINDOW \*win);

int touchline(WINDOW \*win, int start, int count);

int untouchwin(WINDOW \*win);

int wtouchln(WINDOW \*win, int y, int n, int changed);

bool is\_linetouched(WINDOW \*win, int line);

bool is\_wintouched(WINDOW \*win);

#### Description:

touchwin() and touchline() throw away all information about

which parts of the window have been touched, pretending that the

entire window has been drawn on. This is sometimes necessary

when using overlapping windows, since a change to one window

will affect the other window, but the records of which lines

have been changed in the other window will not reflect the

change.

untouchwin() marks all lines in the window as unchanged since

the last call to wrefresh().

wtouchln() makes n lines in the window, starting at line y, look

as if they have (changed == 1) or have not (changed == 0) been

changed since the last call to wrefresh().

is\_linetouched() returns TRUE if the specified line in the

specified window has been changed since the last call to

wrefresh().

is\_wintouched() returns TRUE if the specified window

has been changed since the last call to wrefresh().

Return Value:

All functions return OK on success and ERR on error except

is\_wintouched() and is\_linetouched().

Portability X/Open BSD SYS V

touchwin Y Y Y

touchline Y - 3.0

untouchwin Y - 4.0

wtouchln Y Y Y

is\_linetouched Y - 4.0

is\_wintouched Y - 4.0

--------------------------------------------------------------------------

### util

#### Synopsis:

char \*unctrl(chtype c);

void filter(void);

void use\_env(bool x);

int delay\_output(int ms);

int getcchar(const cchar\_t \*wcval, wchar\_t \*wch, attr\_t \*attrs,

short \*color\_pair, void \*opts);

int setcchar(cchar\_t \*wcval, const wchar\_t \*wch, const attr\_t attrs,

short color\_pair, const void \*opts);

wchar\_t \*wunctrl(cchar\_t \*wc);

int PDC\_mbtowc(wchar\_t \*pwc, const char \*s, size\_t n);

size\_t PDC\_mbstowcs(wchar\_t \*dest, const char \*src, size\_t n);

size\_t PDC\_wcstombs(char \*dest, const wchar\_t \*src, size\_t n);

#### Description:

unctrl() expands the text portion of the chtype c into a

printable string. Control characters are changed to the "^X"

notation; others are passed through. wunctrl() is the wide-

character version of the function.

filter() and use\_env() are no-ops in PDCurses.

delay\_output() inserts an ms millisecond pause in output.

getcchar() works in two modes: When wch is not NULL, it reads

the cchar\_t pointed to by wcval and stores the attributes in

attrs, the color pair in color\_pair, and the text in the

wide-character string wch. When wch is NULL, getcchar() merely

returns the number of wide characters in wcval. In either mode,

the opts argument is unused.

setcchar constructs a cchar\_t at wcval from the wide-character

text at wch, the attributes in attr and the color pair in

color\_pair. The opts argument is unused.

Currently, the length returned by getcchar() is always 1 or 0.

Similarly, setcchar() will only take the first wide character

from wch, and ignore any others that it "should" take (i.e.,

combining characters). Nor will it correctly handle any

character outside the basic multilingual plane (UCS-2).

Return Value:

unctrl() and wunctrl() return NULL on failure. delay\_output()

always returns OK.

getcchar() returns the number of wide characters wcval points to

when wch is NULL; when it's not, getcchar() returns OK or ERR.

setcchar() returns OK or ERR.

Portability X/Open BSD SYS V

unctrl Y Y Y

filter Y - 3.0

use\_env Y - 4.0

delay\_output Y Y Y

getcchar Y

setcchar Y

wunctrl Y

PDC\_mbtowc - - -

PDC\_mbstowcs - - -

PDC\_wcstombs - - -

--------------------------------------------------------------------------

### window

#### Synopsis:

WINDOW \*newwin(int nlines, int ncols, int begy, int begx);

WINDOW \*derwin(WINDOW\* orig, int nlines, int ncols,

int begy, int begx);

WINDOW \*subwin(WINDOW\* orig, int nlines, int ncols,

int begy, int begx);

WINDOW \*dupwin(WINDOW \*win);

int delwin(WINDOW \*win);

int mvwin(WINDOW \*win, int y, int x);

int mvderwin(WINDOW \*win, int pary, int parx);

int syncok(WINDOW \*win, bool bf);

void wsyncup(WINDOW \*win);

void wcursyncup(WINDOW \*win);

void wsyncdown(WINDOW \*win);

WINDOW \*resize\_window(WINDOW \*win, int nlines, int ncols);

int wresize(WINDOW \*win, int nlines, int ncols);

WINDOW \*PDC\_makelines(WINDOW \*win);

WINDOW \*PDC\_makenew(int nlines, int ncols, int begy, int begx);

void PDC\_sync(WINDOW \*win);

#### Description:

newwin() creates a new window with the given number of lines,

nlines and columns, ncols. The upper left corner of the window

is at line begy, column begx. If either nlines or ncols is zero,

they will be defaulted to LINES - begy and COLS - begx. A new

full-screen window is created by calling newwin(0, 0, 0, 0).

delwin() deletes the named window, freeing all memory associated

with it. In the case of overlapping windows, subwindows should

be deleted before the main window.

mvwin() moves the window so that the upper left-hand corner is

at position (y,x). If the move would cause the window to be off

the screen, it is an error and the window is not moved. Moving

subwindows is allowed, but should be avoided. (I don't know why?)

subwin() creates a new sub-window within a window. The

dimensions of the sub-window are nlines lines and ncols columns.

The sub-window is at position (begy, begx) on the screen. This

position is relative to the screen, and not to the window orig.

The sub-window is made in the middle of the window orig, so that

changes made to either window will affect both. When using this

routine, it will often be necessary to call touchwin() before

calling wrefresh().

derwin() is the same as subwin(), except that begy and begx are

relative to the origin of the window orig rather than the

screen. There is no difference between subwindows and derived

windows.

mvderwin() moves a derived window (or subwindow) inside its

parent window. The screen-relative parameters of the window are

not changed. This routine is used to display different parts of

the parent window at the same physical position on the screen.

dupwin() creates an exact duplicate of the window win.

wsyncup() causes a touchwin() of all of the window's parents.

If wsyncok() is called with a second argument of TRUE, this

causes a wsyncup() to be called every time the window is

changed.

wcursyncup() causes the current cursor position of all of a

window's ancestors to reflect the current cursor position of the

current window.

wsyncdown() causes a touchwin() of the current window if any of

its parent's windows have been touched.

resize\_window() allows the user to resize an existing window. It

returns the pointer to the new window, or NULL on failure.

wresize() is an ncurses-compatible wrapper for resize\_window().

Note that, unlike ncurses, it will NOT process any subwindows of

the window. (However, you still can call it \_on\_ subwindows.) It

returns OK or ERR.

PDC\_makenew() allocates all data for a new WINDOW \* except the

actual lines themselves. If it's unable to allocate memory for

the window structure, it will free all allocated memory and

return a NULL pointer.

PDC\_makelines() allocates the memory for the lines.

PDC\_sync() handles wrefresh() and wsyncup() calls when a window

is changed.

#### Return Value:

newwin(), subwin(), derwin() and dupwin() return a pointer

to the new window, or NULL on failure. delwin(), mvwin(),

mvderwin() and syncok() return OK or ERR. wsyncup(),

wcursyncup() and wsyncdown() return nothing.

#### Errors:

It is an error to call resize\_window() before calling initscr().

Also, an error will be generated if we fail to create a newly

sized replacement window for curscr, or stdscr. This could

happen when increasing the window size. NOTE: If this happens,

the previously successfully allocated windows are left alone;

i.e., the resize is NOT cancelled for those windows.

Portability X/Open BSD SYS V

newwin Y Y Y

delwin Y Y Y

mvwin Y Y Y

subwin Y Y Y

derwin Y - Y

mvderwin Y - Y

dupwin Y - 4.0

wsyncup Y - 4.0

syncok Y - 4.0

wcursyncup Y - 4.0

wsyncdown Y - 4.0

resize\_window - - -

wresize - - -

PDC\_makelines - - -

PDC\_makenew - - -

PDC\_sync - - -

--------------------------------------------------------------------------

### clipboard

#### Synopsis:

int PDC\_getclipboard(char \*\*contents, long \*length);

int PDC\_setclipboard(const char \*contents, long length);

int PDC\_freeclipboard(char \*contents);

int PDC\_clearclipboard(void);

#### Description:

PDC\_getclipboard() gets the textual contents of the system's

clipboard. This function returns the contents of the clipboard

in the contents argument. It is the responsibilitiy of the

caller to free the memory returned, via PDC\_freeclipboard().

The length of the clipboard contents is returned in the length

argument.

PDC\_setclipboard copies the supplied text into the system's

clipboard, emptying the clipboard prior to the copy.

PDC\_clearclipboard() clears the internal clipboard.

Return Values:

indicator of success/failure of call.

PDC\_CLIP\_SUCCESS the call was successful

PDC\_CLIP\_MEMORY\_ERROR unable to allocate sufficient memory for

the clipboard contents

PDC\_CLIP\_EMPTY the clipboard contains no text

PDC\_CLIP\_ACCESS\_ERROR no clipboard support

Portability X/Open BSD SYS V

PDC\_getclipboard - - -

PDC\_setclipboard - - -

PDC\_freeclipboard - - -

PDC\_clearclipboard - - -

--------------------------------------------------------------------------

### pdckbd

#### Synopsis:

unsigned long PDC\_get\_input\_fd(void);

#### Description:

PDC\_get\_input\_fd() returns the file descriptor that PDCurses

reads its input from. It can be used for select().

Portability X/Open BSD SYS V

PDC\_get\_input\_fd - - -

--------------------------------------------------------------------------

### pdcsetsc

#### Synopsis:

##### int PDC\_set\_blink(bool blinkon);

##### void PDC\_set\_title(const char \*title);

#### Description:

PDC\_set\_blink() toggles whether the A\_BLINK attribute sets an

actual blink mode (TRUE), or sets the background color to high

intensity (FALSE). The default is platform-dependent (FALSE in

most cases). It returns OK if it could set the state to match

the given parameter, ERR otherwise. Current platforms also

adjust the value of COLORS according to this function -- 16 for

FALSE, and 8 for TRUE.

PDC\_set\_title() sets the title of the window in which the curses

program is running. This function may not do anything on some

platforms. (Currently it only works in Win32 and X11.)

Portability X/Open BSD SYS V

PDC\_set\_blink - - -

PDC\_set\_title - - -

--------------------------------------------------------------------------

### sb

#### Synopsis:

##### int sb\_init(void)

##### int sb\_set\_horz(int total, int viewport, int cur)

##### int sb\_set\_vert(int total, int viewport, int cur)

##### int sb\_get\_horz(int \*total, int \*viewport, int \*cur)

##### int sb\_get\_vert(int \*total, int \*viewport, int \*cur)

##### int sb\_refresh(void);

#### Description:

These functions manipulate the scrollbar.

Return Value:

All functions return OK on success and ERR on error.

Portability X/Open BSD SYS V

sb\_init - - -

sb\_set\_horz - - -

sb\_set\_vert - - -

sb\_get\_horz - - -

sb\_get\_vert - - -

sb\_refresh - - -

--------------------------------------------------------------------------

## X11 Considerations

==================

PDCurses for X11 uses the System V IPC shared memory facility, along

with sockets, to share data between the curses program and the child

process created to manage the X stuff.

When compiling your application, you need to include the <curses.h> or

<xcurses.h> that comes with PDCurses. You also need to link your code

with libXCurses. You may need to link with the following libraries under

X11R5:

Xaw Xmu Xt X11

or, under X11R6:

Xaw Xmu Xt X11 SM ICE Xext

You can run "xcurses-config --libs" to show the link parameters for your

system. If using dynamic linking, on some systems, "-lXCurses" suffices.

By calling Xinitscr() rather than initscr(), you can pass your program

name and resource overrides to PDCurses. The program name is used as the

title of the X window, and for defining X resources specific to your

program.

### Interaction with stdio

----------------------

Be aware that curses programs that expect to have a normal tty

underneath them will be very disappointed! Output directed to stdout

will go to the xterm that invoked the PDCurses application, or to the

console if not invoked directly from an xterm. Similarly, stdin will

expect its input from the same place as stdout.

### X Resources

-----------

PDCurses for X11 recognizes the following resources:

lines

cols

normalFont

italicFont

pointer

pointerForeColor

pointerBackColor

cursorColor

textCursor

colorBlack

colorRed

colorGreen

colorYellow

colorBlue

colorMagenta

colorCyan

colorWhite

colorBoldBlack

colorBoldRed

colorBoldGreen

colorBoldYellow

colorBoldBlue

colorBoldMagenta

colorBoldCyan

colorBoldWhite

bitmap

pixmap

translations

shmmin

borderWidth

borderColor

clickPeriod

doubleClickPeriod

composeKey

lines: Specifies the number of lines the "screen" will have.

Directly equates to LINES.

There is no theoretical maximum.

The minimum value must be 2.

Default: 24

cols: Specifies the number of columns the "screen" will have.

Directly equates to COLS.

There is no theoretical maximum.

The minimum value must be 2.

Default: 80

normalFont: The name of a fixed width font.

Default: 7x13

italicFont: The name of a fixed width font to be used for

characters with A\_ITALIC attributes. Must have the

same cell size as normalFont.

Default: 7x13 (obviously not an italic font)

pointer: The name of a valid pointer cursor.

Default: xterm

pointerForeColor: The foreground color of the pointer.

Default: black

pointerBackColor: The background color of the pointer.

Default: white

textCursor: The alignment of the text cursor; horizontal or vertical.

Default: horizontal

colorBlack: The color of the COLOR\_BLACK attribute.

Default: Black

colorRed: The color of the COLOR\_RED attribute.

Default: red3

colorGreen: The color of the COLOR\_GREEN attribute.

Default: green3

colorYellow: The color of the COLOR\_YELLOW attribute.

Default: yellow3

colorBlue: The color of the COLOR\_BLUE attribute.

Default: blue3

colorMagenta: The color of the COLOR\_MAGENTA attribute.

Default: magenta3

colorCyan: The color of the COLOR\_CYAN attribute.

Default: cyan3

colorWhite: The color of the COLOR\_WHITE attribute.

Default: Grey

colorBoldBlack: COLOR\_BLACK combined with A\_BOLD.

Default: grey40

colorBoldRed: COLOR\_RED combined with A\_BOLD.

Default: red1

colorBoldGreen: COLOR\_GREEN combined with A\_BOLD.

Default: green1

colorBoldYellow: COLOR\_YELLOW combined with A\_BOLD.

Default: yellow1

colorBoldBlue: COLOR\_BLUE combined with A\_BOLD.

Default: blue1

colorBoldMagenta: COLOR\_MAGENTA combined with A\_BOLD.

Default: magenta1

colorBoldCyan: COLOR\_CYAN combined with A\_BOLD.

Default: cyan1

colorBoldWhite: COLOR\_WHITE combined with A\_BOLD.

Default: White

bitmap: The name of a valid bitmap file of depth 1 (black and white)

used for the application's icon. The file is an X bitmap.

Default: a 32x32 or 64x64 pixmap depending on the

window manager

pixmap: The name of a valid pixmap file of any depth

supported by the window manager (color) for the

application's icon, The file is an X11 pixmap. This

resource is only available if the libXpm package has

been installed (most systems have this by default).

This resource overrides the "bitmap" resource.

Default: none, uses default bitmap above

translations: Translations enable the user to customize the action

that occurs when a key, combination of keys, or a

button is pressed. The translations are similar to

those used by xterm.

Defaults:

<Key>: XCursesKeyPress()

<KeyUp>: XCursesKeyPress()

<BtnDown>: XCursesButton()

<BtnUp>: XCursesButton()

<BtnMotion>: XCursesButton()

The most useful action for KeyPress translations is

string(). The argument to the string() action can be

either a string or a hex representation of a

character; e.g., string(0x1b) will send the ASCII

escape character to the application; string("[11~")

will send [ 1 1 ~ , as separate keystrokes.

shmmin: On most systems, there are two Unix kernel parameters

that determine the allowable size of a shared memory

segment. These parameters are usually something like

SHMMIN and SHMMAX. To use shared memory, a program

must allocate a segment of shared memory that is

between these two values. Usually these values are

like 1 for SHMMIN and some large number for SHMMAX.

Sometimes the Unix kernel is configured to have a

value of SHMMIN that is bigger than the size of one

of the shared memory segments that libXCurses uses.

On these systems an error message like:

Cannot allocate shared memory for SCREEN: Invalid argument

will result. To overcome this problem, this resource

should be set to the kernel value for SHMMIN. This

ensures that a shared memory segment will always be

bigger than the kernel value for SHMMIN (and

hopefully less than SHMMAX!)

Default: 0

borderColor: The color of the border around the screen.

Default: black

borderWidth: The width in pixels of the border around the screen.

Default: 0

clickPeriod: The period (in milliseconds) between a button

press and a button release that determines if a click

of a button has occurred.

Default: 100

doubleClickPeriod: The period (in milliseconds) between two button

press events that determines if a double click

of a button has occurred.

Default: 200

composeKey: The name of the X key that defines the "compose key",

which is used to enter characters in the Latin-1

character set above 0xA0. (See "Compose Keys for

Latin-1" below.) This is used only when PDCurses is

built without XIM support. While in compose mode, the

text cursor will appear as a hollow rectangle.

Default: Multi\_key

### Using Resources

---------------

All applications have a top-level class name of "XCurses". If Xinitscr()

is used, it sets an application's top-level widget name. (Otherwise the

name defaults to "PDCurses".)

Examples for app-defaults or .Xdefaults:

!

! resources for XCurses class of programs

!

XCurses\*lines: 30

XCurses\*cols: 80

XCurses\*normalFont: 9x13

XCurses\*bitmap: /tmp/xcurses.xbm

XCurses\*pointer: top\_left\_arrow

!

! resources for testcurs - XCurses

!

testcurs.colorRed: orange

testcurs.colorBlack: midnightblue

testcurs.lines: 25

\*testcurs.Translations: #override \n \

<Key>F12: string(0x1b) string("[11~") \n

!

! resources for THE - XCurses

!

! resources with the \* wildcard can be overridden by a parameter passed

! to initscr()

!

the\*normalFont: 9x15

the\*lines: 40

the\*cols: 86

the\*pointer: xterm

the\*pointerForeColor: white

the\*pointerBackColor: black

!

! resources with the . format can not be overridden by a parameter passed

! to Xinitscr()

!

the.bitmap: /home/mark/the/the64.xbm

the.pixmap: /home/mark/the/the64.xpm

Resources may also be passed as parameters to the Xinitscr() function.

Parameters are strings in the form of switches; e.g., to set the color

"red" to "indianred", and the number of lines to 30, the string passed

to Xinitscr would be: "-colorRed indianred -lines 30"

### Compose Keys for Latin-1

------------------------

When built without XIM support, PDCurses for X11 provides its own,

limited compose key system for Latin-1 characters. The available

combinations are listed here. For a given character, any of the

combinations shown in the last column may be used. To generate a

character, press the "compose" key followed by one of the pairs of

keystrokes. Where no key is evident, the spacebar is used. Thus, to

generate the NO-BREAK SPACE, press the "compose" key followed by two

hits of the spacebar.

With a typical modern X server, you can get many more compose key

combinations by using XIM instead. Configure PDCurses with --enable-xim

to use XIM support.

This document is encoded in UTF-8.

+----+-----+---+---------------------------------+---------------------------+

|Hex | Dec |Chr| Description ISO 10646-1:1993(E) | Compose key combinations |

+----+-----+---+---------------------------------+---------------------------+

| A0 | 160 | | NO-BREAK SPACE | |

| A1 | 161 | Â¡ | INVERTED EXCLAMATION MARK | ! !! |

| A2 | 162 | Â¢ | CENT SIGN | c| |c c/ c$ C$ C| |

| A3 | 163 | Â£ | POUND SIGN | L- L$ L= l- l$ l= |-|

| A4 | 164 | Â¤ | CURRENCY SIGN | xo ox XO g$ |

| A5 | 165 | Â¥ | YEN SIGN | =y y= =Y Y= Y- y$ y-|

| A6 | 166 | Â¦ | BROKEN BAR | | || vb VB |^ |

| A7 | 167 | Â§ | SECTION SIGN | SO SS s! S! so |

| A8 | 168 | Â¨ | DIAERESIS | " "" |

| A9 | 169 | Â© | COPYRIGHT SIGN | CO co OC |

| AA | 170 | Âª | FEMININE ORDINAL INDICATOR | sa SA a\_ A\_ |

| AB | 171 | Â« | LEFT DOUBLE ANGLE QUOTES | << |

| AC | 172 | Â¬ | NOT SIGN | -, no NO |

| AD | 173 | Â­ | SOFT HYPHEN | - -- |

| AE | 174 | Â® | REGISTERED SIGN | RO ro OR |

| AF | 175 | Â¯ | MACRON | -^ \_^ \_\_ |

| B0 | 176 | Â° | DEGREE SIGN | o 0^ 0\* de DE ^0 |

| B1 | 177 | Â± | PLUS-MINUS SIGN | -+ +- |

| B2 | 178 | Â² | SUPERSCRIPT TWO | 2 2^ s2 ^2 |

| B3 | 179 | Â³ | SUPERSCRIPT THREE | 3 3^ s3 ^3 |

| B4 | 180 | Â´ | ACUTE ACCENT | ' '' |

| B5 | 181 | Âµ | MICRO SIGN | u /u /U \*m \*M |

| B6 | 182 | Â¶ | PILCROW SIGN | p! P! pg PG |

| B7 | 183 | Â· | MIDDLE DOT | . .^ .. |

| B8 | 184 | Â¸ | CEDILLA | , ,, |

| B9 | 185 | Â¹ | SUPERSCRIPT ONE | 1 1^ s1 ^1 |

| BA | 186 | Âº | MASCULINE ORDINAL INDICATOR | o\_ s0 S0 |

| BB | 187 | Â» | RIGHT DOUBLE ANGLE QUOTES | >> |

| BC | 188 | Â¼ | VULGAR FRACTION ONE QUARTER | 14 |

| BD | 189 | Â½ | VULGAR FRACTION ONE HALF | 12 |

| BE | 190 | Â¾ | VULGAR FRACTION THREE QUARTERS | 34 |

| BF | 191 | Â¿ | INVERTED QUESTION MARK | ? ?? |

| C0 | 192 | Ã€ | CAPITAL A WITH GRAVE ACCENT | `A A` |

| C1 | 193 | Ã | CAPITAL A WITH ACUTE ACCENT | 'A A' |

| C2 | 194 | Ã‚ | CAPITAL A WITH CIRCUMFLEX ACCENT| ^A A^ A> |

| C3 | 195 | Ãƒ | CAPITAL A WITH TILDE | ~A A~ A- |

| C4 | 196 | Ã„ | CAPITAL A WITH DIAERESIS | "A A" |

| C5 | 197 | Ã… | CAPITAL A WITH RING ABOVE | oA Ao A\* OA \*A |

| C6 | 198 | Ã† | CAPITAL LIGATURE AE | AE |

| C7 | 199 | Ã‡ | CAPITAL C WITH CEDILLA | ,C C, |

| C8 | 200 | Ãˆ | CAPITAL E WITH GRAVE ACCENT | `E E` |

| C9 | 201 | Ã‰ | CAPITAL E WITH ACUTE ACCENT | 'E E' |

| CA | 202 | ÃŠ | CAPITAL E WITH CIRCUMFLEX ACCENT| ^E E^ E> |

| CB | 203 | Ã‹ | CAPITAL E WITH DIAERESIS | "E E" |

| CC | 204 | ÃŒ | CAPITAL I WITH GRAVE ACCENT | `I I` |

| CD | 205 | Ã | CAPITAL I WITH ACUTE ACCENT | 'I I' |

| CE | 206 | ÃŽ | CAPITAL I WITH CIRCUMFLEX ACCENT| ^I I^ I> |

| CF | 207 | Ã | CAPITAL I WITH DIAERESIS | "I I" |

| D0 | 208 | Ã | CAPITAL ETH | D- |

| D1 | 209 | Ã‘ | CAPITAL N WITH TILDE | ~N N~ N- |

| D2 | 210 | Ã’ | CAPITAL O WITH GRAVE ACCENT | `O O` |

| D3 | 211 | Ã“ | CAPITAL O WITH ACUTE ACCENT | 'O O' |

| D4 | 212 | Ã” | CAPITAL O WITH CIRCUMFLEX ACCENT| ^O O^ O> |

| D5 | 213 | Ã• | CAPITAL O WITH TILDE | ~O O~ O- |

| D6 | 214 | Ã– | CAPITAL O WITH DIAERESIS | "O O" |

| D7 | 215 | Ã— | MULTIPLICATION SIGN | x xx XX mu MU |

| D8 | 216 | Ã˜ | CAPITAL O WITH STROKE | /O O/ |

| D9 | 217 | Ã™ | CAPITAL U WITH GRAVE ACCENT | `U U` |

| DA | 218 | Ãš | CAPITAL U WITH ACUTE ACCENT | 'U U' |

| DB | 219 | Ã› | CAPITAL U WITH CIRCUMFLEX ACCENT| ^U U^ U> |

| DC | 220 | Ãœ | CAPITAL U WITH DIAERESIS | "U U" |

| DD | 221 | Ã | CAPITAL Y WITH ACUTE ACCENT | 'Y Y' |

| DE | 222 | Ãž | CAPITAL THORN | P TH |P |

| DF | 223 | ÃŸ | SMALL SHARP S | ss |

| E0 | 224 | Ã  | SMALL A WITH GRAVE ACCENT | `a a` |

| E1 | 225 | Ã¡ | SMALL A WITH ACUTE ACCENT | 'a a' |

| E2 | 226 | Ã¢ | SMALL A WITH CIRCUMFLEX ACCENT | ^a a^ a> |

| E3 | 227 | Ã£ | SMALL A WITH TILDE | ~a a~ a- |

| E4 | 228 | Ã¤ | SMALL A WITH DIAERESIS | "a a" |

| E5 | 229 | Ã¥ | SMALL A WITH RING ABOVE | oa ao Oa a\* \*a |

| E6 | 230 | Ã¦ | SMALL LIGATURE AE | ae |

| E7 | 231 | Ã§ | SMALL C WITH CEDILLA | ,c c, |

| E8 | 232 | Ã¨ | SMALL E WITH GRAVE ACCENT | `e e` |

| E9 | 233 | Ã© | SMALL E WITH ACUTE ACCENT | 'e e' |

| EA | 234 | Ãª | SMALL E WITH CIRCUMFLEX ACCENT | ^e e^ e> |

| EB | 235 | Ã« | SMALL E WITH DIAERESIS | "e e" |

| EC | 236 | Ã¬ | SMALL I WITH GRAVE ACCENT | `i i` |

| ED | 237 | Ã­ | SMALL I WITH ACUTE ACCENT | 'i i' |

| EE | 238 | Ã® | SMALL I WITH CIRCUMFLEX ACCENT | ^i i^ i> |

| EF | 239 | Ã¯ | SMALL I WITH DIAERESIS | "i i" |

| F0 | 240 | Ã° | SMALL ETH | d- |

| F1 | 241 | Ã± | SMALL N WITH TILDE | ~n n~ n- |

| F2 | 242 | Ã² | SMALL O WITH GRAVE ACCENT | `o o` |

| F3 | 243 | Ã³ | SMALL O WITH ACUTE ACCENT | 'o o' |

| F4 | 244 | Ã´ | SMALL O WITH CIRCUMFLEX ACCENT | ^o o^ o> |

| F5 | 245 | Ãµ | SMALL O WITH TILDE | ~o o~ o- |

| F6 | 246 | Ã¶ | SMALL O WITH DIAERESIS | "o o" |

| F7 | 247 | Ã· | DIVISION SIGN | -: :- |

| F8 | 248 | Ã¸ | SMALL O WITH OBLIQUE BAR | /o o/ |

| F9 | 249 | Ã¹ | SMALL U WITH GRAVE ACCENT | `u u` |

| FA | 250 | Ãº | SMALL U WITH ACUTE ACCENT | 'u u' |

| FB | 251 | Ã» | SMALL U WITH CIRCUMFLEX ACCENT | ^u u^ u> |

| FC | 252 | Ã¼ | SMALL U WITH DIAERESIS | "u u" |

| FD | 253 | Ã½ | SMALL Y WITH ACUTE ACCENT | 'y y' |

| FE | 254 | Ã¾ | SMALL THORN | p th |p |

| FF | 255 | Ã¿ | SMALL Y WITH DIAERESIS | "y y" |

+----+-----+---+---------------------------------+---------------------------+

### Deprecated

----------

XCursesProgramName is no longer used. To set the program name, you must

use Xinitscr(), or PDC\_set\_title() to set just the window title.

The XCursesExit() function is now called automatically via atexit().

(Multiple calls to it are OK, so you don't need to remove it if you've

already added it for previous versions of PDCurses.)

XCURSES is no longer defined automatically, but need not be defined,

unless you want the X11-specific prototypes. (Normal curses programs

won't need it.)

--------------------------------------------------------------------------

## SDL Considerations

==================

There are no special requirements to use PDCurses for SDL -- all

PDCurses-compatible code should work fine. (In fact, you can even build

against the Win32 console pdcurses.dll, and then swap in the SDL

pdcurses.dll.) Nothing extra is needed beyond the base SDL library.

However, there are some optional special features, described here.

The principal limitation of this port is that input is currently

restricted to ASCII (i.e., 0-127), plus the special keys like KEY\_LEFT.

(You could have Unicode input, but then the input wouldn't match the

output, which is in Code Page 437.) Also, see the note about the

potential for incomplete output under "PDC\_update\_rects()", below.

### Fonts

-----

The font is a simple BMP, 32 characters wide by 8 characters tall,

preferably with a palette. (BMPs without palettes still work, but in

that case, no attributes will be available, nor will the cursor work.)

The first entry in the palette (usually black) is treated as the

background color; the last entry (usually white) is treated as the

foreground. These are changed or made transparent as appropriate; any

other colors in the palette are passed through unchanged. So -- although

a one-bit depth is sufficient for a normal font -- you could redraw some

characters as multi-colored tiles.

The font must be monospaced. The size of each character is derived by

dividing the width of the BMP by 32 and the height by 8. There is no

constraint on the dimensions.

As provided in the default font and expected by acs\_map[], the font is

in Code Page 437 form. But you can of course use any layout if you're

not relying on correct values for the ACS\_\* macros.

The font can be set via the environment variable PDC\_FONT. If it's not

set, PDCurses looks for a file named "pdcfont.bmp" in the current

directory at the time of initscr(). If neither is found, it uses the

built-in default font encoded in deffont.h.

### Backgrounds

-----------

PDCurses for SDL supports an optional background image BMP. This is used

whenever start\_color() has not been called (see the ptest demo for an

example), or when use\_default\_colors() has been called after

start\_color(), and the background color of a pair has been set to -1

(see newdemo, worm, and rain for examples). The usage parallels that of

ncurses in an appropriate terminal (e.g., Gnome Terminal). The image is

tiled to cover the PDCurses window, and can be any size or depth.

As with the font, you can point to a location for the background via the

environment variable PDC\_BACKGROUND; "pdcback.bmp" is the fallback.

(There is no default background.)

### Icons

-----

The icon (used with SDL\_WM\_SetIcon() -- not used for the executable

file) can be set via the environment variable PDC\_ICON, and falls back

to "pdcicon.bmp", and then to the built-in icon from deficon.h. The

built-in icon is the PDCurses logo, as seen in ../x11/little\_icon.xbm.

The SDL docs say that the icon must be 32x32, at least for use with MS

Windows.

If pdc\_screen is preinitialized (see below), PDCurses does not attempt

to set the icon.

### Screen size

-----------

The default screen size is 80x25 characters (whatever size they may be),

but you can override this via the environment variables PDC\_COLS and/or

PDC\_LINES. (Some other ports use COLS and LINES; this is not done here

because those values are, or should be, those of the controlling

terminal, and PDCurses for SDL is independent of the terminal.) If

pdc\_screen is preinitialized (see below), these are ignored.

### Integration with SDL

--------------------

If you want to go further, you can mix PDCurses and SDL functions. (Of

course this is extremely non-portable!) To aid you, there are several

external variables and functions specific to the SDL port; you could

include pdcsdl.h, or just add the declarations you need in your code:

PDCEX SDL\_Surface \*pdc\_screen, \*pdc\_font, \*pdc\_icon, \*pdc\_back;

PDCEX int pdc\_sheight, pdc\_swidth, pdc\_yoffset, pdc\_xoffset;

void PDC\_update\_rects(void);

void PDC\_retile(void);

pdc\_screen is the main surface, created by SDL\_SetVideoMode(), unless

it's preset before initscr(). You can perform normal SDL operations on

this surface, but PDCurses won't respect them when it updates. (For

that, see PDC\_retile().) As an alternative, you can preinitialize this

surface before calling initscr(). In that case, you can use pdc\_sheight,

pdc\_swidth, pdc\_yoffset and/or pdc\_xoffset (q.v.) to confine PDCurses to

only a specific area of the surface, reserving the rest for other SDL

operations. If you preinitialize pdc\_screen, you'll have to close it

yourself; PDCurses will ignore resize events, and won't try to set the

icon. Also note that if you preinitialize pdc\_screen, it need not be the

display surface.

pdc\_font, pdc\_icon, and pdc\_back are the SDL\_surfaces for the font,

icon, and background, respectively. You can set any or all of them

before initscr(), and thus override any of the other ways to set them.

But note that pdc\_icon will be ignored if pdc\_screen is preset.

pdc\_sheight and pdc\_swidth are the dimensions of the area of pdc\_screen

to be used by PDCurses. You can preset them before initscr(); if either

is not set, it defaults to the full screen size minus the x or y offset,

as appropriate.

pdc\_xoffset and pdc\_yoffset are the x and y offset for the area of

pdc\_screen to be used by PDCurses. See the sdltest demo for an example.

PDC\_retile() makes a copy of pdc\_screen, then tiles it with the

background image, if any. The resulting surface is used as the

background for transparent character cells. PDC\_retile() is called from

initscr() and resize\_term(). However, you can also use it at other

times, to take advantage of the way it copies pdc\_screen: Draw some SDL

stuff; call PDC\_retile(); do some curses stuff -- it will use whatever

was on pdc\_screen as the background. Then you can erase the curses

screen, do some more SDL stuff, and call PDC\_retile() again to make a

new background. (If you don't erase the curses screen, it will be

incorporated into the background when you call PDC\_retile().) But this

only works if no background image is set.

PDC\_update\_rects() is how the screen actually gets updated. For

performance reasons, when drawing, PDCurses for SDL maintains a table of

rectangles that need updating, and only updates (by calling this

function) during getch(), napms(), or when the table gets full.

Normally, this is sufficient; but if you're pausing in some way other

than by using napms(), and you're not doing keyboard checks, you may get

an incomplete update. If that happens, you can call PDC\_update\_rects()

manually.

### Interaction with stdio

----------------------

As with X11, it's a bad idea to mix curses and stdio calls. (In fact,

that's true for PDCurses on any platform; but especially these two,

which don't run under terminals.) Depending on how SDL is built, stdout

and stderr may be redirected to files.