# Contents

## Preface
11

### Curses Library Functions
17
- `addch(3XCURSES)` 18
- `addchstr(3XCURSES)` 20
- `addnstr(3XCURSES)` 21
- `addnwstr(3XCURSES)` 23
- `add_wch(3XCURSES)` 25
- `add_wchnstr(3XCURSES)` 27
- `attr_get(3XCURSES)` 29
- `attroff(3XCURSES)` 31
- `baudrate(3XCURSES)` 32
- `beep(3XCURSES)` 33
- `bkgd(3XCURSES)` 34
- `bkgnd(3XCURSES)` 36
- `border(3XCURSES)` 38
- `border_set(3XCURSES)` 40
- `can_change_color(3XCURSES)` 42
- `cbreak(3XCURSES)` 45
- `chgat(3XCURSES)` 46
- `clear(3XCURSES)` 47
- `clearok(3XCURSES)` 48
- `clrtobot(3XCURSES)` 50
- `clrtoeol(3XCURSES)` 51
- `COLS(3XCURSES)` 52
curs_move(3CURSES) 137

curs_outopts(3CURSES) 138

curs_overlay(3CURSES) 141

curs_pad(3CURSES) 142

curs_printw(3CURSES) 144

curs_refresh(3CURSES) 145

curs_scanw(3CURSES) 147

curs_scr_dump(3CURSES) 148

curs_scroll(3CURSES) 150

curs_set(3XCURSES) 151

curs_slk(3CURSES) 152

curs_termattrs(3CURSES) 154

curs_termcap(3CURSES) 156

curs_terminfo(3CURSES) 158

curs_touch(3CURSES) 162

curs_util(3CURSES) 164

curs_window(3CURSES) 166

cur_term(3XCURSES) 169

def_prog_mode(3XCURSES) 170

delay_output(3XCURSES) 171

delch(3XCURSES) 172

del_curterm(3XCURSES) 173

deleteln(3XCURSES) 175

delscreen(3XCURSES) 176

delwin(3XCURSES) 177

derwin(3XCURSES) 178

doupdate(3XCURSES) 180

dupwin(3XCURSES) 181

echo(3XCURSES) 182

echochar(3XCURSES) 183

echo_wchar(3XCURSES) 184

endwin(3XCURSES) 185

erasechar(3XCURSES) 186

filter(3XCURSES) 187

flushinp(3XCURSES) 188

form_cursor(3CURSES) 189

form_data(3CURSES) 190
form_driver(3CURSES) 191
form_field(3CURSES) 194
form_field_attributes(3CURSES) 195
form_field_buffer(3CURSES) 196
form_field_info(3CURSES) 197
form_field_just(3CURSES) 198
form_field_new(3CURSES) 199
form_field_opts(3CURSES) 200
form_field_type(3CURSES) 202
form_field_userptr(3CURSES) 204
form_field_validation(3CURSES) 205
form_hook(3CURSES) 206
form_new(3CURSES) 208
form_new_page(3CURSES) 209
form_opts(3CURSES) 210
form_page(3CURSES) 211
form_post(3CURSES) 213
forms(3CURSES) 214
form_userptr(3CURSES) 218
form_win(3CURSES) 219
getbegyx(3XCURSES) 220
getcchar(3XCURSES) 221
getch(3XCURSES) 222
getnstr(3XCURSES) 227
getn_wstr(3XCURSES) 229
get_wch(3XCURSES) 230
getwin(3XCURSES) 232
halfdelay(3XCURSES) 233
has_ic(3XCURSES) 234
hline(3XCURSES) 235
hline_set(3XCURSES) 236
idcok(3XCURSES) 237
immedok(3XCURSES) 238
inch(3XCURSES) 239
inchnstr(3XCURSES) 240
innstr(3XCURSES) 242

innwstr(3XCURSES) 245
insch(3XCURSES) 247
insdelln(3XCURSES) 248
insertln(3XCURSES) 249
insnstr(3XCURSES) 250
ins_nwstr(3XCURSES) 252
ins_wch(3XCURSES) 254
intrflush(3XCURSES) 255
in_wch(3XCURSES) 256
in_wchnstr(3XCURSES) 257
is_linetouched(3XCURSES) 259
keyname(3XCURSES) 261
keypad(3XCURSES) 262
LINES(3XCURSES) 266
longname(3XCURSES) 267
menu_attributes(3CURSES) 268
menu_cursor(3CURSES) 270
menu_driver(3CURSES) 271
menu_format(3CURSES) 273
menu_hook(3CURSES) 274
menu_item_current(3CURSES) 276
menu_item_name(3CURSES) 278
menu_item_new(3CURSES) 279
menu_item_opts(3CURSES) 280
menu_items(3CURSES) 281
menu_item_userptr(3CURSES) 282
menu_item_value(3CURSES) 283
menu_item_visible(3CURSES) 284
menu_mark(3CURSES) 285
menu_new(3CURSES) 286
menu_opts(3CURSES) 287
menu_pattern(3CURSES) 288
menu_post(3CURSES) 289
menus(3CURSES) 290
menu_userptr(3CURSES) 294
menu_win(3CURSES) 295
meta(3XCURSES) 296
Preface

Both novice users and those familiar with the SunOS operating system can use online man pages to obtain information about the system and its features. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

Overview

The following contains a brief description of each man page section and the information it references:

- Section 1 describes, in alphabetical order, commands available with the operating system.
- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.
- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.
- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2.
- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.
- Section 5 contains miscellaneous documentation such as character-set tables.
- Section 6 contains available games and demos.
- Section 7 describes various special files that refer to specific hardware peripherals and device drivers. STREAMS software drivers, modules and the STREAMS-generic set of system calls are also described.
- Section 9 provides reference information needed to write device drivers in the kernel environment. It describes two device driver interface specifications: the Device Driver Interface (DDI) and the Driver/Kernel Interface (DKI).
- Section 9E describes the DDI/DKI, DDI-only, and DKI-only entry-point routines a developer can include in a device driver.
- Section 9F describes the kernel functions available for use by device drivers.
- Section 9S describes the data structures used by drivers to share information between the driver and the kernel.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the intro pages for more information and detail about each section, and man(1) for more information about man pages in general.

NAME

This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS

This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full path name is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ] Brackets. The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.

... Ellipses. Several values can be provided for the previous argument, or the previous argument can be specified multiple times, for example, "filename ...".

| Separator. Only one of the arguments separated by this character can be specified at a time.

{ } Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.
PROTOCOL  
This section occurs only in subsection 3R to indicate the protocol description file.

DESCRIPTION  
This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES. Interactive commands, subcommands, requests, macros, and functions are described under USAGE.

IOCTL  
This section appears on pages in Section 7 only. Only the device class that supplies appropriate parameters to the ioctl(2) system call is called ioctl and generates its own heading. ioctl calls for a specific device are listed alphabetically (on the man page for that specific device). ioctl calls are used for a particular class of devices all of which have an io ending, such as mti(7I).

OPTIONS  
This section lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.

OPERANDS  
This section lists the command operands and describes how they affect the actions of the command.

OUTPUT  
This section describes the output – standard output, standard error, or output files – generated by the command.

RETURN VALUES  
If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.

ERRORS  
On failure, most functions place an error code in the global variable errno indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than
one condition can cause the same error, each condition is described in a separate paragraph under the error code.

**USAGE**

This section lists special rules, features, and commands that require in-depth explanations. The subsections listed here are used to explain built-in functionality:

- Commands
- Modifiers
- Variables
- Expressions
- Input Grammar

**EXAMPLES**

This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command-line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%`, or if the user must be superuser, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS, and USAGE sections.

**ENVIRONMENT VARIABLES**

This section lists any environment variables that the command or function affects, followed by a brief description of the effect.

**EXIT STATUS**

This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion, and values other than zero for various error conditions.

**FILES**

This section lists all file names referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.

**ATTRIBUTES**

This section lists characteristics of commands, utilities, and device drivers by defining the attribute type and its corresponding value. See `attributes(5)` for more information.

**SEE ALSO**

This section lists references to other man pages, in-house documentation, and outside publications.
DIAGNOSTICS

This section lists diagnostic messages with a brief explanation of the condition causing the error.

WARNINGS

This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.

NOTES

This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.

BUGS

This section describes known bugs and, wherever possible, suggests workarounds.
Curses Library Functions
addch(3XCURSES)

NAME  addch, mvaddch, mvwaddch, waddch – add a character (with rendition) to a window

SYNOPSIS

```c
#include <curses.h>

int addch(const chtype ch);
int mvaddch(int y, int x, const chtype ch);
int mvwaddch(WINDOW *win, int y, int x, const chtype ch);
int waddch(WINDOW *win, const chtype ch);
```

DESCRIPTION

The `addch()` function writes a character to the `stdscr` window at the current cursor position. The `mvaddch()` and `mvwaddch()` functions write the character to the position indicated by the `x` (column) and `y` (row) parameters. The `mvaddch()` function writes the character to the `stdscr` window, while `mvwaddch()` writes the character to the window specified by `win`. The `waddch()` function is identical to `addch()`, but writes the character to the window specified by `win`.

These functions advance the cursor after writing the character. Characters that do not fit on the end of the current line are wrapped to the beginning of the next line unless the current line is the last line of the window and scrolling is disabled. In that situation, characters which extend beyond the end of the line are discarded.

When `ch` is a backspace, carriage return, newline, or tab, X/Open Curses moves the cursor appropriately. Each tab character moves the cursor to the next tab stop. By default, tab stops occur every eight columns. When `ch` is a control character other than backspace, carriage return, newline, or tab, it is written using `^x` notation, where `x` is a printable character. When X/Open Curses writes `ch` to the last character position on a line, it automatically generates a newline. When `ch` is written to the last character position of a scrolling region and `scrollok()` is enabled, X/Open Curses scrolls the scrolling region up one line (see `clearok(3XCURSES)`).

PARAMETERS

- `wchstr` Is a pointer to the `cchar_t` string to be copied to the window.
- `n` Is the maximum number of characters to be copied from `wchstr`. If `n` is less than 0, the entire string is written or as much of it as fits on the line.
- `y` Is the `y` (row) coordinate of the starting position of `wchstr` in the window.
- `x` Is the `x` (column) coordinate of the starting position of `wchstr` in the window.
- `win` Is a pointer to the window to which the string is to be copied.

RETURN VALUES

On success, these functions return `OK`. Otherwise, they return `ERR`.

ERRORS

None.
SEE ALSO

attroff(3XCURSES), bkgdset(3XCURSES), douupdate(3XCURSES),
inch(3XCURSES), insch(3XCURSES), nl(3XCURSES), printw(3XCURSES),
scrollok(3XCURSES), scrl(3XCURSES), terminfo(4)
addchstr(3XCURSES)

NAME  addchstr, addchnstr, mvaddchstr, mvaddchnstr, mvwaddchstr, mvwaddchnstr, waddchstr, waddchnstr – copy a character string (with renditions) to a window

SYNOPSIS

#include <curses.h>

int addchstr(const chtype *chstr);
int addchnstr(const chtype *chstr, int n);
int mvaddchnstr(int y, int x, const chtype *chstr, int n);
int mvwaddchnstr(int y, int x, const chtype *chstr);
int mvwaddchnstr(WINDOW *win, int y, int x, const chtype *chstr, int n);
int mvwaddchnstr(WINDOW *win, int y, int x, const chtype *chstr);
int waddchstr(WINDOW *win, const chtype *chstr);
int waddchnstr(WINDOW *win, const chtype *chstr, int n);

DESCRIPTION

The addchstr() function copies the chtype character string to the stdscr window at the current cursor position. The mvaddchnstr() and mvwaddchnstr() functions copy the character string to the starting position indicated by the x (column) and y (row) parameters (the former to the stdscr window; the latter to window win). The waddchstr() is identical to addchstr(), but writes to the window specified by win.

The addchnstr(), waddchnstr(), mvaddchnstr(), and mvwaddchnstr() functions write n characters to the window, or as many as will fit on the line. If n is less than 0, the entire string is written, or as much of it as fits on the line. The former two functions place the string at the current cursor position; the latter two commands use the position specified by the x and y parameters.

These functions differ from the addstr(3XCURSES) set of functions in two important respects. First, these functions do not advance the cursor after writing the string to the window. Second, the current window rendition is not combined with the character; only the attributes that are already part of the chtype character are used.

PARAMETERS

chstr  Is a pointer to the chtype string to be copied to the window.

n  Is the maximum number of characters to be copied from chstr. If n is less than 0, the entire string is written or as much of it as fits on the line.

y  Is the y (row) coordinate of the starting position of chstr in the window.

x  Is the x (column) coordinate of the starting position of chstr in the window.

win  Is a pointer to the window to which the string is to be copied.

RETURN VALUES

On success, these functions return OK. Otherwise, they return ERR.

ERRORS

None.

SEE ALSO

addch(3XCURSES), addnstr(3XCURSES), attroff(3XCURSES)
addnstr, addstr, mvaddnstr, mvaddstr, mvwaddnstr, mvwaddstr, waddnstr, waddstr – add a multi-byte character string (without rendition) to a window

#include <curses.h>

int addnstr(const char *str, int n);
int addstr(const char *str);
int mvaddnstr(int y, int x, const char *str, int n);
int mvaddstr(int y, int x, const char *str);
int mvwaddnstr(WINDOW *win, int y, int x, const char *str, int n);
int mvwaddstr(WINDOW *win, int y, int x, const char *str);
int waddstr(WINDOW *win, const char *str);
int waddnstr(WINDOW *win, const char *str, int n);

DESCRIPTION
The addstr() function writes a null-terminated string of multi-byte characters to the stdscr window at the current cursor position. The waddstr() function performs an identical action, but writes the character to the window specified by win. The mvaddstr() and mvwaddstr() functions write the string to the position indicated by the x (column) and y (row) parameters (the former to the stdscr window; the latter to window win).

The addnstr(), waddnstr(), mvaddnstr(), and mvwaddnstr() functions are similar but write at most n characters to the window. If n is less than 0, the entire string is written.

All of these functions advance the cursor after writing the string.

These functions are functionally equivalent to calling the corresponding function from the addch(3XCURSES) set of functions once for each character in the string. Refer to the curses(3XCURSES) man page for a complete description of special character handling and of the interaction between the window rendition (or background character and rendition) and the character written.

Note that these functions differ from the addchstr() set of functions in that the addchstr(3XCURSES) functions copy the string as is (without combining each character with the window rendition or the background character and rendition).

PARAMETERS
str    Is a pointer to the character string that is to be written to the window.

n      Is the maximum number of characters to be copied from str. If n is less than 0, the entire string is written or as much of it as fits on the line.

y      Is the y (row) coordinate of the starting position of str in the window.

x      Is the x (column) coordinate of the starting position of str in the window.

win    Is a pointer to the window in which the string is to be written.
addnstr(3XCURSES)

**RETURN VALUES**
On success, these functions return `OK`. Otherwise, they return `ERR`.

**ERRORS**
None.

**SEE ALSO**
addch(3XCURSES), addchstr(3XCURSES), curses(3XCURSES)
addnwstr(3XCURSES)

NAME
addnwstr, addwstr, mvaddnwstr, mvaddwstr, mvwaddnwstr, mvwaddwstr,
waddnwstr, waddwstr – add a wide-character string to a window

SYNOPSIS
#include <curses.h>

int addnwstr(const wchar_t *wstr, int n);
int addwstr(const wchar_t *wstr);
int mvaddnwstr(int y, int x, const wchar_t *wstr, int n);
int mvaddwstr(int y, int x, const wchar_t *wstr);
int mvwaddnwstr(WINDOW*win, int y, int x, const wchar_t *wstr, int n);
int mvwaddwstr(WINDOW*win, int y, int x, const wchar_t *wstr);
int waddnwstr(WINDOW*win, const wchar_t *wstr, int n);
int waddwstr(WINDOW*win, const wchar_t *wstr);

DESCRIPTION
The addwstr() function writes a null-terminated wide-character string to the
stdscr window at the current cursor position. The waddwstr() function performs
an identical action, but writes the string to the window specified by win. The
mvaddwstr() and mvwaddwstr() functions write the string to the position
indicated by the x (column) and y (row) parameters (the former to the stdscr
window; the latter to window win).

The addnwstr(), waddnwstr(), mvaddnwstr(), and mvwaddnwstr() functions
write at most n characters to the window. If n is less than 0, the entire string is written.
The former two functions place the characters at the current cursor position; the latter
two commands use the position specified by the x and y parameters.

All of these functions advance the cursor after writing the string.

These functions are functionally equivalent to building a cchar_t from the wchar_t
and the window rendition (or background character and rendition) and calling the
wadd_wch(3XCURSES) function once for each wchar_t in the string. Refer to the
curses(3XCURSES) man page for a complete description of special character
handling and of the interaction between the window rendition (or background
character and rendition) and the character written.

Note that these functions differ from the add_wchnstr(3XCURSES) set of functions in
that the latter copy the string as is (without combining each character with the
foreground and background attributes of the window).

PARAMETERS

wstr Is a pointer to the wide-character string that is to be written to the window.

n Is the maximum number of characters to be copied from wstr. If n is less
than 0, the entire string is written or as much of it as fits on the line.

y Is the y (row) coordinate of the starting position of wstr in the window.
addnwstr(3XCURSES)

\[ x \quad \text{Is the x (column) coordinate of the starting position of } wstr \text{ in the window.} \]

\[ win \quad \text{Is a pointer to the window in which the string is to be written.} \]

**RETURN VALUES**

On success, these functions return **OK**. Otherwise, they return **ERR**.

**ERRORS**

None.

**SEE ALSO**

add_wch(3XCURSES), add_wchnstr(3XCURSES), curses(3XCURSES)
### add_wch(3XCURSES)

**NAME**
add_wch, mvadd_wch, mvwadd_wch, wadd_wch – add a complex character (with rendition) to a window

**SYNOPSIS**
```c
#include <curses.h>

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);
```

**DESCRIPTION**
The `add_wch()` function writes a complex character to the `stdscr` window at the current cursor position. The `mvadd_wch()` and `mvwadd_wch()` functions write the character to the position indicated by the `x` (column) and `y` (row) parameters. The `mvadd_wch()` function writes the character to the `stdscr` window, while `mvwadd_wch()` writes the character to the window specified by `win`. The `wadd_wch()` function is identical to `add_wch()`, but writes the character to the window specified by `win`. These functions advance the cursor after writing the character.

If `wch` is a spacing complex character, X/Open Curses replaces any previous character at the specified location with `wch` (and its rendition). If `wch` is a non-spacing complex character, X/Open Curses preserves all existing characters at the specified location and adds the non-spacing characters of `wch` to the spacing complex character. It ignores the rendition associated with `wch`.

Characters that do not fit on the end of the current line are wrapped to the beginning of the next line unless the current line is the last line of the window and scrolling is disabled. In that situation, X/Open Curses discards characters which extend beyond the end of the line.

When `wch` is a backspace, carriage return, newline, or tab, X/Open Curses moves the cursor appropriately as described in the `curses(3XCURSES)` man page. Each tab character moves the cursor to the next tab stop. By default, tab stops occur every eight columns. When `wch` is a control character other than a backspace, carriage return, newline, or tab, it is written using `^x` notation, where `x` is a printable character. When X/Open Curses writes `wch` to the last character position on a line, it automatically generates a newline. When `wch` is written to the last character position of a scrolling region and `scrollok()` is enabled, X/Open Curses scrolls the scrolling region up one line (see `clearok(3XCURSES)`).

**PARAMETERS**
- `wch` Is the character/attribute pair (rendition) to be written to the window.
- `win` Is a pointer to the window in which the character is to be written.
- `y` Is the `y` (row) coordinate of the character’s position in the window.
- `x` Is the `x` (column) coordinate of the character’s position in the window.

**RETURN VALUES**
On success, these functions return `OK`. Otherwise, they return `ERR`.

---

Curses Library Functions  25
add_wch(3XCURSES)

ERRORS | None.

SEE ALSO | attr_off(3XCURSES), bgkndset(3XCURSES), curses(3XCURSES),
doupdate(3XCURSES), in_wch(3XCURSES), ins_wch(3XCURSES), nl(3XCURSES),
printw(3XCURSES), scrollok(3XCURSES), scroll(3XCURSES),
setscrreg(3XCURSES), terminfo(4)
add_wchnstr(3XCURSES)

NAME
add_wchnstr, add_wchstr, mvadd_wchnstr, mvadd_wchstr, mvwadd_wchnstr,
mvadd_wchstr, wadd_wchnstr, wadd_wchstr – copy a string of complex characters
(with renditions) to a window

SYNOPSIS
#include <curses.h>

int add_wchnstr(const cchar_t *wchstr, int n);
int add_wchstr(const cchar_t *wchstr);
int mvadd_wchnstr(int y, int x, const cchar_t *wchstr, int n);
int mvadd_wchstr(int y, int x, const cchar_t *wchstr);
int mvwadd_wchnstr(WINDOW *win, int y, int x, const cchar_t *wchstr, int n);
int mvwaddchstr(WINDOW *win, int y, int x, const cchar_t *wchstr);
int wadd_wchstr(WINDOW *win, const cchar_t *wchstr);
int wadd_wchnstr(WINDOW *win, const cchar_t *wchstr, int n);

DESCRIPTION
The add_wchstr() function copies the string of cchar_t characters to the stdscr
window at the current cursor position. The mvadd_wchstr() and
mvwadd_wchstr() functions copy the string to the starting position indicated by the
x (column) and y (row) parameters (the former to the stdscr window; the latter to
window win). The wadd_wchstr() is identical to add_wchstr(), but writes to the
window specified by win.

The add_wchnstr(), wadd_wchnstr(), mvadd_wchnstr(), and
mvwadd_wchnstr() functions write n characters to the window, or as many as will
fit on the line. If n is less than 0, the entire string is written, or as much of it as fits on
the line. The former two functions place the string at the current cursor position; the
latter two commands use the position specified by the x and y parameters.

These functions differ from the addwstr(3XCURSES) set of functions in two
important respects. First, these functions do not advance the cursor after writing the
string to the window. Second, the current window rendition (that is, the combination
of attributes and color pair) is not combined with the character; only those attributes
that are already part of the cchar_t character are used.

PARAMETERS
wchstr Is a pointer to the cchar_t string to be copied to the window.

n Is the maximum number of characters to be copied from wchstr. If
n is less than 0, the entire string is written or as much of it as fits on
the line.

y Is the y (row) coordinate of the starting position of wchstr in the
window.

x Is the x (column) coordinate of the starting position of wchstr in the
window.
add_wchnstr(3XCURSES)

| win | Is a pointer to the window to which the string is to be copied. |

RETURN VALUES

On success, these functions return OK. Otherwise, they return ERR.

ERRORS

None.

SEE ALSO

addnwstr(3XCURSES), add_wch(3XCURSES), attr_off(3XCURSES)
NAME | attr_get, attr_off, attr_on, attr_set, color_set, wattr_get, wattr_off, wattr_on, wattr_set, wcolor_set – control window attributes
SYNOPSIS | #include <curses.h>
  int attr_get(attr_t *attrs, short *color, 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, void *opts);
  int color_set(short *color, void *opts);
  int wattr_get(WINDOW *win, attr_t attrs, short *color, 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, void *opts);
  int wcolor_set(WINDOW *win, short color, void *opts);

DESCRIPTION | The attr_get() function retrieves the current rendition of stdscr. The wattr_get() function retrieves the current rendition of window win. If attrs or color is a null pointer, no information is retrieved.

The attr_off() and attr_on() functions unset and set, respectively, the specified window attributes of stdscr. These functions only affect the attributes specified; attributes that existed before the call are retained.

The wattr_off() and wattr_on() functions unset or set the specified attributes for window win.

The attr_set() and wattr_set() functions change the rendition of stdscr and win; the old values are not retained.

The color_set() and wcolor_set() functions set the window color of stdscr and win to color.

The attributes and color pairs that can be used are specified in the Attributes, Color Pairs, and Renditions section of the curses(3XCURSES) man page.

PARAMETERS | 
attrs | Is a pointer to the foreground window attributes to be set or unset.
color | Is a pointer to a color pair number.
opts | Is reserved for future use.
win | Is a pointer to the window in which attribute changes are to be made.

RETURN VALUES | These functions always return OK.

ERRORS | None.
attr_get(3XCURSES)

SEE ALSO  add_wch(3XCURSES), addnstr(3XCURSES), attroff(3XCURSES),
           bkgrndset(3XCURSES), curses(3XCURSES), init_color(3XCURSES),
           start_color(3XCURSES)
attron(), attron(), attrset(), wattroff(), wattron(), wattrset() – change foreground window attributes

#include <curses.h>

int attroff(int attrs);
int attron(int attrs);
int attrset(int attrs);
int wattroff(WINDOW *win, int attrs);
int wattron(WINDOW *win, int attrs);
int wattrset(WINDOW *win, int attrs);

The attroff() and attron() functions unset and set, respectively, the specified window attributes of stdscr. These functions only affect the attributes specified; attributes that existed before the call are retained. The wattroff() and wattron() functions unset or set the specified attributes for window win.

The attrset() and wattrset() functions change the specified window renditions of stdscr and win to new values; the old values are not retained.

The attributes that can be used are specified in the Attributes, Color Pairs, and Renditions section of the curses(3XCURSES) man page.

Here is an example that prints some text using the current window rendition, adds underlining, changes the attributes, prints more text, then changes the attributes back.

printw("This word is");
attron(A_UNDERLINE);
printw("underlined.");
attroff(A_NORMAL);
printw("This is back to normal text.
");
refresh( );

PARAMETERS

attrs are the foreground window attributes to be set or unset.

win Is a pointer to the window in which attribute changes are to be made.

RETURN VALUES

These functions always return OK or 1.

ERRORS
None.

USAGE
All of these functions may be macros.

SEE ALSO
addch(3XCURSES), addnstr(3XCURSES), attr_get(3XCURSES), bkgdset(3XCURSES), curses(3XCURSES), init_color(3XCURSES), start_color(3XCURSES)
### NAME
baudrate – return terminal baud rate

### SYNOPSIS
```c
#include <curses.h>

int baudrate(void);
```

### DESCRIPTION
The `baudrate()` function returns the terminal’s data communication line and output speed in bits per second (for example, 9600).

### RETURN VALUES
The `baudrate()` function returns the output speed of the terminal.

### ERRORS
None.
NAME | beep, flash – activate audio-visual alarm
SYNOPSIS | #include <curses.h>
           | int beep(void);
           | int flash(void);
DESCRIPTION | The beep() and flash() functions produce an audio and visual alarm on the
terminal, respectively. If the terminal has the capability, beep() sounds a bell or beep
and flash() flashes the screen. One alarm is substituted for another if the terminal
does not support the capability called (see terminfo(4) bel and flash capabilities).
For example, a call to beep() for a terminal without that capability results in a flash.
RETURN VALUES | These functions always return OK.
ERRORS | None.
SEE ALSO | terminfo(4)
NAME
bkgd, bkgdset, getbkgd, wbkgd, wbkgdset – set or get the background character (and rendition) of window

SYNOPSIS
#include <curses.h>

int bkgd(chtype ch);
void bkgdset(chtype ch);
chtpe getbkgd(WINDOW *win);
int wbkgd(WINDOW *win, chtype ch);
void wbkgdset(WINDOW *win, chtype ch);

DESCRIPTION
The bkgdset() and wbkgdset() functions turn off the previous background attributes, logical OR the requested attributes into the window rendition, and set the background property of the current or specified window based on the information in ch. If ch refers to a multi-column character, the results are undefined.

The bkgd() and wbkgd() functions turn off the previous background attributes, logical OR the requested attributes into the window rendition, and set the background property of the current or specified window and then apply this setting to every character position in that window:

- The rendition of every character on the screen is changed to the new window rendition.
- Wherever the former background character appears, it is changed to the new background character.

The getbkgd() function extracts the specified window’s background character and rendition.

PARAMETERS
ch
Is the background character to be set.

win
Is a pointer to the window in which the background character is to be set.

RETURN VALUES
Upon successful completion, the bkgd() and wbkgd() functions return OK. Otherwise, they return ERR.

The bkgdset() and wbkgdset() functions do not return a value.

Upon successful completion, the getbkgd() function returns the specified window’s background character and rendition. Otherwise, it returns (chtpe) ERR.

ERRORS
No errors are defined.

USAGE
These functions are only guaranteed to operate reliably on character sets in which each character fits into a single byte, whose attributes can be expressed using only constants with the A_ prefix.
SEE ALSO
addch(3XCURSES), addchstr(3XCURSES), attroff(3XCURSES),
bkgrnd(3XCURSES), clear(3XCURSES), clrtoeol(3XCURSES),
clrtobot(3XCURSES), erase(3XCURSES), inch(3XCURSES),
mvprintw(3XCURSES)
**NAME**
bkgrnd, bkgrndset, getbkgrnd, wbkgrnd, wbkgrndset, wgetbkgrnd – set or get the background character (and rendition) of window using a complex character

**SYNOPSIS**
```
#include <curses.h>
int bkgrnd(const cchar_t *wch);
void bkgrndset(const cchar_t *wch);
int getbkgrnd(cchar_t *wch);
nint wbkgrnd(WINDOW *win, const cchar_t *wch);
void wbkgrndset(WINDOW *win, const cchar_t *wch);
int wgetbkgrnd(WINDOW *win, cchar_t *wch);
```

**DESCRIPTION**
The **bkgrndset()** and **wbkgrndset()** functions turn off the previous background attributes, logical OR the requested attributes into the window rendition, and set the background property of the current or specified window based on the information in **wch**.

The **bkgrnd()** and **wbkgrnd()** functions turn off the previous background attributes, logical OR the requested attributes into the window rendition, and set the background property of the current or specified window and then apply this setting to every character position in that window:

- The rendition of every character on the screen is changed to the new window rendition.
- Wherever the former background character appears, it is changed to the new background character.

If **wch** refers to a non-spacing complex character for **bkgrnd()**, **bkgrndset()**, **wbkgrnd()**, and **wbkgrndset()**, then **wch** is added to the existing spacing complex character that is the background character. If **wch** refers to a multi-column character, the results are unspecified.

The **getbkgrnd()** and **wgetbkgrnd()** functions store, into the area pointed to buy **wch**, the window's background character and rendition.

**PARAMETERS**
- **wch** Is a pointer to the complex background character to be set.
- **win** Is a pointer to the window in which the complex background character is to be set.

**RETURN VALUES**
The **bkgrndset()** and **wbkgrndset()** functions do not return a value.

Upon successful completion, the other functions return **OK**. Otherwise, they return **ERR**.

**ERRORS**
No errors are defined.
SEE ALSO

add_wch(3XCURSES), add_wchnstr(3XCURSES), addch(3XCURSES),
addchstr(3XCURSES), attroff(3XCURSES), bkgd(3XCURSES),
clear(3XCURSES), clrtoeol(3XCURSES), clrtobot(3XCURSES),
erase(3XCURSES), inch(3XCURSES), mvprintw(3XCURSES)
border(3XCURSES)

NAME  border, box, wborder – add a single-byte border to a window

SYNOPSIS

#include <curses.h>

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);

DESCRIPTION

The border() and wborder() functions draw a border around the specified
window. All parameters must be single-byte characters whose rendition can be
expressed using only constants beginning with ACS_. A parameter with the value of 0
is replaced by the default value.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Constant</th>
<th>Default Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>verch</td>
<td>ACS_VLINE</td>
<td></td>
</tr>
<tr>
<td>horch</td>
<td>ACS_HLINE</td>
<td>-</td>
</tr>
<tr>
<td>ls</td>
<td>ACS_VLINE</td>
<td></td>
</tr>
<tr>
<td>rs</td>
<td>ACS_VLINE</td>
<td></td>
</tr>
<tr>
<td>ts</td>
<td>ACS_HLINE</td>
<td>-</td>
</tr>
<tr>
<td>bs</td>
<td>ACS_HLINE</td>
<td>-</td>
</tr>
<tr>
<td>bl</td>
<td>ACS_BLCORNER</td>
<td>+</td>
</tr>
<tr>
<td>br</td>
<td>ACS_BRCORNER</td>
<td>+</td>
</tr>
<tr>
<td>tl</td>
<td>ACS_ULCORNER</td>
<td>+</td>
</tr>
<tr>
<td>tr</td>
<td>ACS_URCORNER</td>
<td>+</td>
</tr>
</tbody>
</table>

The call

box(win, verch, horch) is a short form for

wborder(win, verch, verch, horch, horch, 0, 0, 0, 0)

When the window is boxed, the bottom and top rows and right and left columns
overwrite existing text.

PARAMETERS

ls          Is the character and rendition used for the left side of the border.
<table>
<thead>
<tr>
<th>rs</th>
<th>Is the character and rendition used for the right side of the border.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ts</td>
<td>Is the character and rendition used for the top of the border.</td>
</tr>
<tr>
<td>bs</td>
<td>Is the character and rendition used for the bottom of the border.</td>
</tr>
<tr>
<td>tl</td>
<td>Is the character and rendition used for the top-left corner of the</td>
</tr>
<tr>
<td></td>
<td>border.</td>
</tr>
<tr>
<td>tr</td>
<td>Is the character and rendition used for the top-right corner of the</td>
</tr>
<tr>
<td></td>
<td>border.</td>
</tr>
<tr>
<td>bl</td>
<td>Is the character and rendition used for the bottom-left corner of</td>
</tr>
<tr>
<td></td>
<td>the border.</td>
</tr>
<tr>
<td>br</td>
<td>Is the character and rendition used for the bottom-right corner of</td>
</tr>
<tr>
<td></td>
<td>the border.</td>
</tr>
<tr>
<td>win</td>
<td>Is the pointer to the window in which the border or box is to be</td>
</tr>
<tr>
<td></td>
<td>drawn.</td>
</tr>
<tr>
<td>verch</td>
<td>Is the character and rendition used for the left and right columns</td>
</tr>
<tr>
<td></td>
<td>of the box.</td>
</tr>
<tr>
<td>horch</td>
<td>Is the character and rendition used for the top and bottom rows of</td>
</tr>
<tr>
<td></td>
<td>the box.</td>
</tr>
</tbody>
</table>

**RETURN VALUES**  
On success, these functions return `OK`. Otherwise, they return `ERR`.  

**ERRORS**  
None.  

**SEE ALSO**  
`add_wch(3XCURSES)`, `addch(3XCURSES)`, `attr_get(3XCURSES)`, `attroff(3XCURSES)`, `border_set(3XCURSES)`
NAME  
border_set, box_set, wborder_set -- use complex characters (and renditions) to draw borders

SYNOPSIS  
#include <curses.h>

int border_set(const cchar_t *ls, const cchar_t *rs, const cchar_t *ls, 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 *ls, 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);

DESCRIPTION  
The border_set() and wborder_set() functions draw a border around the specified window. All parameters must be spacing complex characters with renditions. A parameter which is a null pointer is replaced by the default character.

Constant Values for Borders

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Constant</th>
<th>Default Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>verch</td>
<td>WACS_VLINE</td>
<td></td>
</tr>
<tr>
<td>horch</td>
<td>WACS_HLINE</td>
<td>-</td>
</tr>
<tr>
<td>ls</td>
<td>WACS_VLINE</td>
<td></td>
</tr>
<tr>
<td>rs</td>
<td>WACS_VLINE</td>
<td></td>
</tr>
<tr>
<td>ts</td>
<td>WACS_HLINE</td>
<td>-</td>
</tr>
<tr>
<td>bs</td>
<td>WACS_HLINE</td>
<td>-</td>
</tr>
<tr>
<td>bl</td>
<td>WACS_BLCORNER</td>
<td>+</td>
</tr>
<tr>
<td>br</td>
<td>WACS_BRCORNER</td>
<td>+</td>
</tr>
<tr>
<td>tl</td>
<td>WACS_ULCORNER</td>
<td>+</td>
</tr>
<tr>
<td>tr</td>
<td>WACS_URCORNER</td>
<td>+</td>
</tr>
</tbody>
</table>

The call

box_set(win,  
verch, horch) is a short form for

wborder_set(win,  
verch, verch,  
horch, horch, NULL,
When the window is boxed, the bottom and top rows and right and left columns are unavailable for text.

**PARAMETERS**

- **ls**: Is the character and rendition used for the left side of the border.
- **rs**: Is the character and rendition used for the right side of the border.
- **ts**: Is the character and rendition used for the top of the border.
- **bs**: Is the character and rendition used for the bottom of the border.
- **tl**: Is the character and rendition used for the top-left corner of the border.
- **tr**: Is the character and rendition used for the top-right corner of the border.
- **bl**: Is the character and rendition used for the bottom-left corner of the border.
- **br**: Is the character and rendition used for the bottom-right corner of the border.
- **win**: Is the pointer to the window in which the border or box is to be drawn.
- **verch**: Is the character and rendition used for the left and right columns of the box.
- **horch**: Is the character and rendition used for the top and bottom rows of the box.

**RETURN VALUES**

On success, these functions return `OK`. Otherwise, they return `ERR`.

**ERRORS**

None.

**SEE ALSO**

- `add_wch(3XCURSES)`, `addch(3XCURSES)`, `attr_get(3XCURSES)`, `attroff(3XCURSES)`, `border(3XCURSES)`
These functions manipulate color on terminals that support color.

### Querying Capabilities
The `has_colors()` function indicates whether the terminal is a color terminal. The `can_change_color()` function indicates whether the terminal is a color terminal on which colors can be redefined.

### Initialization
The `start_color()` function must be called to enable use of colors and before any color manipulation function is called. The function initializes eight basic colors (black, red, green, yellow, blue, magenta, cyan, and white) that can be specified by the color macros (such as `COLOR_BLACK`) defined in `<curses.h>`. The initial appearance of these colors is unspecified.

The function also initializes two global external variables:

- `COLORS` defines the number of colors that the terminal supports. See Color Identification below. If `COLORS` is 0, the terminal does not support redefinition of colors and `can_change_color()` will return `FALSE`.
- `COLOR_PAIRS` defines the maximum number of color-pairs that the terminal supports. See User-defined Color Pairs below.

The `start_color()` function also restores the colors on the terminal to terminal-specific initial values. The initial background color is assumed to be black for all terminals.

### Color Identification
The `init_color()` function redefines color number `color`, on terminals that support the redefinition of colors, to have the red, green, and blue intensity components

### User-defined Color Pairs
The `init_pair()` function redefines color pair `pair`, on terminals that support the redefinition of color pairs, to have the red, green, and blue intensity components
User-defined Color Pairs

Specified by `red`, `green`, and `blue`, respectively. Calling `init_color()` also changes all occurrences of the specified color on the screen to the new definition.

The `color_content()` function identifies the intensity components of color number `color`. It stores the red, green, and blue intensity components of this color in the addresses pointed to by `red`, `green`, and `blue`, respectively.

For both functions, the `color` argument must be in the range from 0 to and including `COLORS-1`. Valid intensity value range from 0 (no intensity component) up to and including 1000 (maximum intensity in that component).

Calling `init_pair()` defines or redefines color-pair number `pair` to have foreground color `f` and background color `b`. Calling `init_pair()` changes any characters that were displayed in the color pair’s old definition to the new definition and refreshes the screen.

After defining the color pair, the macro `COLOR_PAIR(n)` returns the value of color pair `n`. This value is the color attribute as it would be extracted from a `chtype`. Controversy, the macro `COLOR_NUMBER(value)` returns the color pair number associated with the color attribute `value`.

The `pair_content()` retrieves the component colors of a color-pair number `pair`. It stores the foreground and background color numbers in the variables pointed to by `f` and `b`, respectively.

With `init_pair()` and `pair_content()`, the value of `pair` must be in a range from 0 to and including `COLOR_PAIRS-1`. Valid values for `f` and `b` are the range from 0 to and including `COLORS-1`.

**PARAMETERS**

| `color` | Is the number of the color for which to provide information (0 to `COLORS-1`).
| `red` | Is a pointer to the RGB value for the amount of red in `color`.
| `green` | Is a pointer to the RGB value for the amount of green in `color`.
| `blue` | Is a pointer to the RGB value for the amount of blue in `color`.
| `n` | Is the number of a color pair.
| `pair` | Is the number of the color pair for which to provide information (1 to `COLOR_PAIRS-1`).
| `f` | Is a pointer to the number of the foreground color (0 to `COLORS-1`) in `pair`.
| `b` | Is a pointer to the number of the background color (0 to `COLORS-1`) in `pair`.
| `value` | Is a color attribute value.

**RETURN VALUES**

The `has_colors()` function returns `TRUE` if the terminal can manipulate colors. Otherwise, it returns `FALSE`.  

Curses Library Functions 43
The `can_change_color()` function returns `TRUE` if the terminal supports colors and is able to change their definitions. Otherwise, it returns `FALSE`.

Upon successful completion, the other functions return `OK`. Otherwise, they return `ERR`.

**ERRORS**
No errors are defined.

**USAGE**
To use these functions, `start_color()` must be called, usually right after `initscr(3XCURSES)`.

The `can_change_color()` and `has_colors()` functions facilitate writing terminal-independent applications. For example, a programmer can use them to decide whether to use color or some other video attribute.

On color terminals, a typical value of `COLORS` is 8 and the macros such as `COLOR_BLACK` return a value within the range from 0 to and including 7. However, applications cannot rely on this to be true.

**SEE ALSO**
`attroff(3XCURSES), delscreen(3XCURSES), initscr(3XCURSES)`
cbreak(3XCURSES)

NAME
cbreak, nocbreak, noraw, raw – set input mode controls

SYNOPSIS
#include <curses.h>

int cbreak(void);
int nocbreak(void);
int noraw(void);
int raw(void);

DESCRIPTION
The cbreak() function enables the character input mode. This overrides any previous call to the raw() function and turns the stty flag ICANON off.

The nocbreak() function sets the line canonical mode and turns the stty flag ICANON on without touching the ISIG or IXON flags.

The noraw() function sets the line canonical mode and turns the the stty flags ICANON, ISIG, and IXON all on.

The raw() function sets the character input mode and turns the stty flags ICANON, ISIG, and IXON all off. This mode provides maximum control over input.

It is important to remember that the terminal may or may not be in character mode operation initially. Most interactive programs require cbreak() to be enabled.

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None.

SEE ALSO
getch(3XCURSES), halfdelay(3XCURSES), nodelay(3XCURSES),
timeout(3XCURSES), termio(7I)
NAME
chgat, mvchgat, mvwchgat, wchgat – change the rendition of characters in a window

SYNOPSIS
#include <curses.h>

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);

DESCRIPTION
These functions change the renditions of the next \( n \) characters in the current or
specified window (or of the remaining characters on the current or specified line, if \( n \)
is \( -1 \)), beginning at the current or specified cursor position. The attributes and colors
are specified by \( attr \) and \( color \) as for setcchar(3XCURSES).

These function neither update the cursor nor perform wrapping. A value of \( n \) that is
greater than the remaining characters on a line is not an error.

The \( opts \) argument is reserved for definition in a future release. Currently, the
application must provide a null pointer for \( opts \).

PARAMETERS
\( n \) Is the number of characters whose rendition is to be changed.
\( attr \) Is the set of attributes to be assigned to the characters.
\( color \) Is the new color pair to be assigned to the characters.
\( opts \) Is reserved for future use. Currently, this must be a null pointer.
\( y \) Is the \( y \) (row) coordinate of the starting position in the window.
\( x \) Is the \( x \) (column) coordinate of the starting position in the window.
\( win \) Is a pointer to the window in which the rendition of characters is
to be changed.

RETURN VALUES
Upon successful completion, these functions returned OK. Otherwise, they return
ERR.

ERRORS
No errors are defined.

SEE ALSO
bkgrnd(3XCURSES), setcchar(3XCURSES)
NAME | clear, erase, wclear, werase – clear a window
SYNOPSIS | `#include <curses.h>

int clear(void);
int erase(void);
int wclear(WINDOW *win);
int werase(WINDOW *win);

DESCRIPTION | The `clear()` and `erase()` functions clear `stdscr`, destroying its previous contents. The `wclear()` and `werase()` functions perform the same action, but clear the window specified by `win` instead of `stdscr`.

The `clear()` and `wclear()` functions also call the `clearok()` function. This function clears and redraws the entire screen on the next call to `refresh(3XCURSES)` or `wrefresh(3XCURSES)` for the window.

The current background character (and attributes) is used to clear the screen.

PARAMETERS | `win` Is a pointer to the window that is to be cleared.
ERRORS | `OK` Successful completion.
`ERR` An error occurred.

SEE ALSO | `bkgdset(3XCURSES)`, `clearok(3XCURSES)`, `clrtobot(3XCURSES)`, `clrtoeol(3XCURSES)`, `doupdate(3XCURSES)`, `refresh(3XCURSES)`, `wrefresh(3XCURSES)`
These functions set options that deal with the output within Curses functions.

The `clearok()` function assigns the value of `bf` to an internal flag in the specified window that governs clearing of the screen during a refresh. If, during a refresh operation on the specified window, the flag in `curscr` is TRUE or the flag in the specified window is TRUE, `clearok()` clears the screen, redraws it in its entirety, and sets the flag to FALSE in `curscr` and in the specified window. The initial state is unspecified.

The `idlok()` function specifies whether the implementation may use the hardware insert-line, delete-line, and scroll features of terminals so equipped. If `bf` is TRUE, use of these features is enabled. If `bf` is FALSE, use of these features is disabled and lines are instead redrawn as required. The initial state is FALSE.

The `leaveok()` function controls the cursor position after a refresh operation. If `bf` is TRUE, refresh operations on the specified window may leave the terminal’s cursor at an arbitrary position. If `bf` is FALSE, then at the end of any refresh operation, the terminal’s cursor is positioned at the cursor position contained in the specified window. The initial state is FALSE.

The `scrollok()` function controls the use of scrolling. If `bf` is TRUE, then scrolling is enabled for the specified window. If `bf` is FALSE, scrolling is disabled for the specified window. The initial state is FALSE.

The `setscrreg()` and `wsetscrreg()` functions define a software scrolling region in the current or specified window. The `top` and `bottom` arguments are the line numbers of the first and last line defining 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 last line of the margin causes all lines in the scrolling region to scroll one line in the direction of the first line. Only characters in the window are scrolled. If a software scrolling region is set and `scrollok()` is not enabled, an attempt to move off the last line of the margin does not reposition any lines in the scrolling region.

PARAMETERS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>win</code></td>
<td>Is a pointer to a window.</td>
</tr>
<tr>
<td><code>bf</code></td>
<td>Is a Boolean expression.</td>
</tr>
</tbody>
</table>
Is the top line of the scrolling region (top of the window is line 0).
bot Is the bottom line of the scrolling region (top of the window is line 0).

RETURN VALUES
Upon successful completion, the `setscrreg()` and `wsetscrreg()` functions return `OK`. Otherwise, they return `ERR`.

The other functions always return `OK`.

ERRORS
No errors are defined.

USAGE
The only reason to enable the `idlok()` feature is to use scrolling to achieve the visual effect of motion of a partial window, such as for a screen editor. In other cases, the feature can be visually annoying.

The `leaveok()` option provides greater efficiency for applications that do not use the cursor.

SEE ALSO
`bkgdset(3XCURSES)`, `clear(3XCURSES)`, `doupdate(3XCURSES)`, `scrl(3XCURSES)`
**NAME**  
clrtobot, wclrtobot – clear to the end of a window

**SYNOPSIS**  
```c
#include <curses.h>

int clrtobot(void);
int wclrtobot(WINDOW *win);
```

**DESCRIPTION**  
The `clrtobot()` function clears all characters in the `stdscr` window from the cursor to the end of the window. The `wclrtobot()` function performs the same action in the window specified by `win` instead of in `stdscr`. The current background character (and rendition) is used to clear the screen.

If the clearing action results in clearing only a portion of a multicolumn character, background characters are displayed in place of the remaining portion.

**PARAMETERS**  
`win` Is a pointer to the window that is to be cleared.

**RETURN VALUES**  
On success, these functions return OK. Otherwise, they return ERR.

**ERRORS**  
None.

**SEE ALSO**  
bkgdset(3XCURSES), clear(3XCURSES), clearok(3XCURSES), crltoeol(3XCURSES)
NAME  clrtoeol, wclrtoeol – clear to the end of a line

SYNOPSIS
#include <curses.h>

int clrtoeol(void);
int wclrtoeol(WINDOW *win);

DESCRIPTION
The clrtoeol() function clears the current line from the cursor to the right margin in the stdscr window. The wclrtoeol() function performs the same action, but in the window specified by win instead of stdscr. The current background character (and rendition) is used to clear the screen.

If the clearing action results in clearing only a portion of a multicolumn character, background characters are displayed in place of the remaining portion.

PARAMETERS
win          Is a pointer to the window in which to clear to the end of the line.

RETURN VALUES
On success, these functions return OK. Otherwise, they return FALSE.

ERRORS
None.

SEE ALSO
bkgdset(3XCURSES), clear(3XCURSES), clearok(3XCURSES), clrtobot(3XCURSES)
<table>
<thead>
<tr>
<th>NAME</th>
<th>COLS – number of columns on terminal screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>#include &lt;curses.h&gt;</td>
</tr>
<tr>
<td></td>
<td>extern int COLS;</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The external variable COLS indicates the number of columns on the terminal screen.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>initscr(3XCURSES)</td>
</tr>
</tbody>
</table>
#copywin

## NAME
copywin — overlay or overwrite any portion of window

## SYNOPSIS
```
#include <curses.h>

int copywin(const WINDOW *srcwin, WINDOW *dstwin, int sminrow, int smincol, int dminrow, int dmincol, int dmaxrow, int dmaxcol, int overlay);
```

## PARAMETERS
- **srcwin**: Is a pointer to the source window to be copied.
- **dstwin**: Is a pointer to the destination window to be overlayed or overwritten.
- **sminrow**: Is the row coordinate of the upper left corner of the rectangular area on the source window to be copied.
- **smincol**: Is the column coordinate of the upper left corner of the rectangular area on the source window to be copied.
- **dminrow**: Is the row coordinate of the upper left corner of the rectangular area on the destination window to be overlayed or overwritten.
- **dmincol**: Is the column coordinate of the upper left corner of the rectangular area on destination window to be overlayed or overwritten.
- **dmaxrow**: Is the row coordinate of the lower right corner of the rectangular area on the destination window to be overlayed or overwritten.
- **dmaxcol**: Is the column coordinate of the lower right corner of the rectangular area on the destination window to be overlayed or overwritten.
- **overlay**: Is a TRUE or FALSE value that determines whether the destination window is overlayed or overwritten.

## DESCRIPTION
The `copywin()` function provides a finer granularity of control over the `overlay(3XCURSES)` and `overwrite(3XCURSES)` functions. As in the `prefresh()` function (see `newpad(3XCURSES)`), a rectangle is specified in the destination window, `(dminrow, dmincol)` and `(dmaxrow, dmaxcol)`, and the upper-left-corner coordinates of the source window, `(smincol, sminrow)`. If `overlay` is TRUE, then copying is non-destructive, as in `overlay()`. If `overlay` is FALSE, then copying is destructive, as in `overwrite()`.

## RETURN VALUES
Upon successful completion, the `copywin()` function returns OK. Otherwise, it returns ERR.

## ERRORS
No errors are defined.

## SEE ALSO
`curses(3XCURSES), newpad(3XCURSES), overlay(3XCURSES)`
With the addch(), waddch(), mvaddch(), and mvwaddch() routines, the character \( ch \) is put into the window at the current cursor position of the window and the position of the window cursor is advanced. Its function is similar to that of putchar(). At the right margin, an automatic newline is performed. At the bottom of the scrolling region, if scrollok() is enabled, the scrolling region is scrolled up one line.

If \( ch \) is a tab, newline, or backspace, the cursor is moved appropriately within the window. A newline also does a clrtoeol() before moving. Tabs are considered to be at every eighth column. If \( ch \) is another control character, it is drawn in the \(^X\) notation. Calling winch() after adding a control character does not return the control character, but instead returns the representation of the control character. See curs_inch(3CURSES).

Video attributes can be combined with a character by OR-ing them into the parameter. This results in these attributes also being set. (The intent here is that text, including attributes, can be copied from one place to another using inch() and addch().) (see standout(), predefined video attribute constants, on the curs_attr(3CURSES) page).

The echochar() and wechochar() routines are functionally equivalent to a call to addch() followed by a call to refresh(), or a call to waddch followed by a call to wrefresh(). The knowledge that only a single character is being output is taken into consideration and, for non-control characters, a considerable performance gain might be seen by using these routines instead of their equivalents.

The following variables may be used to add line drawing characters to the screen with routines of the addch() family. When variables are defined for the terminal, the A_ALTCHARSET bit is turned on (see curs_attr(3CURSES)). Otherwise, the default character listed below is stored in the variable. The names chosen are consistent with the VT100 nomenclature.
### curs_addch(3CURSES)

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Glyph Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACS_ULCORNER</td>
<td>+</td>
<td>upper left-hand corner</td>
</tr>
<tr>
<td>ACS_LLCORNER</td>
<td>+</td>
<td>lower left-hand corner</td>
</tr>
<tr>
<td>ACS_URCORNER</td>
<td>+</td>
<td>upper right-hand corner</td>
</tr>
<tr>
<td>ACS_LRCORNER</td>
<td>+</td>
<td>lower right-hand corner</td>
</tr>
<tr>
<td>ACS_RTEE</td>
<td>+</td>
<td>right tee</td>
</tr>
<tr>
<td>ACS_LTEE</td>
<td>+</td>
<td>left tee</td>
</tr>
<tr>
<td>ACS_BTEE</td>
<td>+</td>
<td>bottom tee</td>
</tr>
<tr>
<td>ACS_TTEE</td>
<td>+</td>
<td>top tee</td>
</tr>
<tr>
<td>ACS_HLINE</td>
<td>-</td>
<td>horizontal line</td>
</tr>
<tr>
<td>ACS_VLINE</td>
<td></td>
<td>vertical line</td>
</tr>
<tr>
<td>ACS_PLUS</td>
<td>+</td>
<td>plus</td>
</tr>
<tr>
<td>ACS_S1</td>
<td>-</td>
<td>scan line 1</td>
</tr>
<tr>
<td>ACS_S9</td>
<td>-</td>
<td>scan line 9</td>
</tr>
<tr>
<td>ACS_DIAMOND</td>
<td>+</td>
<td>diamond</td>
</tr>
<tr>
<td>ACS_CKBOARD</td>
<td>:</td>
<td>checker board (stipple)</td>
</tr>
<tr>
<td>ACS_DEGREE</td>
<td>’</td>
<td>degree symbol</td>
</tr>
<tr>
<td>ACS_PLMINUS</td>
<td>#</td>
<td>plus/minus</td>
</tr>
<tr>
<td>ACS_BULLET</td>
<td>o</td>
<td>bullet</td>
</tr>
<tr>
<td>ACS_LARROW</td>
<td>&lt;</td>
<td>arrow pointing left</td>
</tr>
<tr>
<td>ACS_RARROW</td>
<td>&gt;</td>
<td>arrow pointing right</td>
</tr>
<tr>
<td>ACS_DARROW</td>
<td>v</td>
<td>arrow pointing down</td>
</tr>
<tr>
<td>ACS_UARROW</td>
<td>^</td>
<td>arrow pointing up</td>
</tr>
<tr>
<td>ACS_BOARD</td>
<td>#</td>
<td>board of squares</td>
</tr>
<tr>
<td>ACS_LANTERN</td>
<td>#</td>
<td>lantern symbol</td>
</tr>
<tr>
<td>ACS_BLOCK</td>
<td>#</td>
<td>solid square block</td>
</tr>
</tbody>
</table>

### RETURN VALUES

All routines return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion.

### ATTRIBUTES

See attributes(5) for descriptions of the following attributes:
curs_addch(3CURSES)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

curs_attr(3CURSES), curs_clear(3CURSES), curs_inch(3CURSES),
curs_outopts(3CURSES), curs_refresh(3CURSES), curses(3CURSES),
putc(3), attributes(5)

NOTES

The header <curses.h> automatically includes the headers <stdio.h> and
<unctrl.h>.

Note that addch(), mvaddch(), mvwaddch(), and echochar() may be macros.
curs_addchstr(3CURSES)

NAME
curs_addchstr, addchstr, addchnstr, waddchstr, waddchnstr, mvaddchstr,
mvaddchnstr, mvwaddchstr, mvwaddchnstr – add string of characters and attributes to a curses window

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int addchstr(chtype *chstr);
int addchnstr(chtype *chstr, int n);
int waddchstr(WINDOW *win, chtype *chstr);
int waddchnstr(WINDOW *win, chtype *chstr, int n);
int mvaddchstr(int y, int x, chtype *chstr);
int mvaddchnstr(int y, int x, chtype *chstr, int n);
int mvwaddchstr(WINDOW *win, int y, int x, chtype *chstr);
int mvwaddchnstr(WINDOW *win, int y, int x, chtype *chstr, int n);

DESCRIPTION
All of these routines copy chstr directly into the window image structure starting at the current cursor 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, to the maximum number that fit on the line.

The position of the window cursor is not advanced. These routines works faster than waddnstr() (see curs_addstr(3CURSES)) because they merely copy chstr into the window image structure. On the other hand, care must be taken when using these functions because they do not perform any kind of checking (such as for the newline character), they do not advance the current cursor position, and they truncate the string, rather then wrapping it around to the next line.

RETURN VALUES
All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_addstr(3CURSES), curses(3CURSES), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.

Note that all routines except waddchnstr() and waddchstr() may be macros.
curs_addstr(3CURSES)

NAME  
curs_addstr, addstr, addnstr, waddstr, waddnstr, mvaddstr, mvaddnstr, mvwaddstr, 
  mvwaddnstr – add a string of characters to a curses window and advance cursor

SYNOPSIS  
cc [... flag ...] file [... -lcurses [...] 
#include <curses.h>
int addstr(char *str);
int addnstr(char *str, int n);
int waddstr(WINDOW *win, char *str);
int waddnstr(WINDOW *win, char *str, int n);
int mvaddstr(int y, int x, char *str);
int mvaddnstr(int y, int x, char *str, int n);
int mvwaddstr(WINDOW *win, int y, int x, char *str);
int mvwaddnstr(WINDOW *win, int y, int x, char *str, int n);

DESCRIPTION  
All of these routines write all the characters of the null terminated character string str 
on the given window. It is similar to calling waddch() once for each character in the 
string. The four routines with n as the last argument write at most n characters. If n is 
negative, then the entire string will be added.

RETURN VALUES  
All routines return the integer ERR upon failure and an integer value other than ERR. 
upon successful completion.

ATTRIBUTES  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curs_addch(3CURSES), curses(3CURSES), attributes(5)

NOTES  
The header <curses.h> automatically includes the headers <stdio.h> and 
<unctrl.h>.

Note that all routines except waddstr() and waddnstr() may not be macros.
NAME  
curs_addwch, addwch, waddwch, mvaddwch, mvwaddwch, echowchar, wechowchar

SYNOPSIS  
c  c  [flag...]  file...  -l  curses  [library...]
 #include<curses.h>
 int addwch(chtype wch);
 int waddwch(WINDOW *win, chtype wch);
 int mvaddwch(int y, int x, chtype wch);
 int mvwaddwch(WINDOW *win, int y, int x, chtype wch);
 int echowchar(chtype wch);
 int wechowchar(WINDOW *win, chtype wch);

DESCRIPTION  
The addwch(), waddwch(), mvaddwch(), and mvwaddwch() routines put the
character wch, holding a wchar_t character, into the window at the current cursor
position of the window and advance the position of the window cursor. Their function
is similar to that of putwchar(3C) in the C multibyte library. At the right margin, an
automatic newline is performed. At the bottom of the scrolling region, if scrollok is
enabled, the scrolling region is scrolled up one line.

If wch is a tab, newline, or backspace, the cursor is moved appropriately within the
window. A newline also does a clrtoeol(3CURSES) before moving. Tabs are
considered to be at every eighth column. If wch is another control character, it is drawn
in the ^X notation. Calling winwch(3CURSES) after adding a control character does
not return the control character, but instead returns the representation of the control
character.

Video attributes can be combined with a wchar_t character by OR-ing them into the
parameter. This results in these attributes also being set. (The intent here is that text,
including attributes, can be copied from one place to another using inwch() and
addwch().) See standout(3CURSES), predefined video attribute constants.

The echowchar() and wechowchar() routines are functionally equivalent to a call
to addwch() followed by a call to refresh(3CURSES), or a call to waddwch() followed by a call to wrefresh(3CURSES). The knowledge that only a single
character is being output is taken into consideration and, for non-control characters, a
considerable performance gain might be seen by using these routines instead of their
equivalents.

Line Graphics  
The following variables may be used to add line drawing characters to the screen with
routines of the addwch() family. When variables are defined for the terminal, the
A_ALTCHARSET bit is turned on. (See curs_attr(3CURSES)). Otherwise, the default
character listed below is stored in the variable. The names chosen are consistent with
the VT100 nomenclature.
curs_addwch(3CURSES)

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Glyph Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACS_ULCORNER</td>
<td>+</td>
<td>upper left-hand corner</td>
</tr>
<tr>
<td>ACS_LLCORNER</td>
<td>+</td>
<td>lower left-hand corner</td>
</tr>
<tr>
<td>ACS_URCORNER</td>
<td>+</td>
<td>upper right-hand corner</td>
</tr>
<tr>
<td>ACS_LRCORNER</td>
<td>+</td>
<td>lower right-hand corner</td>
</tr>
<tr>
<td>ACS_RTEE</td>
<td>+</td>
<td>right tee</td>
</tr>
<tr>
<td>ACS_LTEE</td>
<td>+</td>
<td>left tee</td>
</tr>
<tr>
<td>ACS_BTEE</td>
<td>+</td>
<td>bottom tee</td>
</tr>
<tr>
<td>ACS_TTEE</td>
<td>+</td>
<td>top tee</td>
</tr>
<tr>
<td>ACS_HLINE</td>
<td>-</td>
<td>horizontal line</td>
</tr>
<tr>
<td>ACS_VLINE</td>
<td>|</td>
<td>vertical line</td>
</tr>
<tr>
<td>ACS_PLUS</td>
<td>+</td>
<td>plus</td>
</tr>
<tr>
<td>ACS_S1</td>
<td>-</td>
<td>scan line 1</td>
</tr>
<tr>
<td>ACS_S9</td>
<td>-</td>
<td>scan line 9</td>
</tr>
<tr>
<td>ACS_DIAMOND</td>
<td>+</td>
<td>diamond</td>
</tr>
<tr>
<td>ACS_CKBOARD</td>
<td>:</td>
<td>checker board (stipple)</td>
</tr>
</tbody>
</table>
| ACS_DEGREE          | \^
    | degree symbol         |
| ACS_PLMINUS         | #       | plus/minus                 |
| ACS_BULLET          | o       | bullet                     |
| ACS_LARROW          | <       | arrow pointing left        |
| ACS_RARROW          | >       | arrow pointing right       |
| ACS_DARROW          | \n      | arrow pointing down        |
| ACS_UARROW          | \n      | arrow pointing up          |
| ACS_BOARD           | #       | board of squares           |
| ACS_LANTERN         | #       | lantern symbol             |
| ACS_BLOCK           | #       | solid square block         |

RETURN VALUE
All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion, unless otherwise noted in the preceding routine descriptions.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:
curs_addwch(3CURSES)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
putwch(3C), clrtoeol(3CURSES), curses(3CURSES), curs_attr(3CURSES),
curs_inwch(3CURSES), curs_outopts(3CURSES), refresh(3CURSES),
standout(3CURSES), winwch(3CURSES), wrefresh(3CURSES), attributes(5)

NOTES
The header file <curses.h> automatically includes the header files <stdio.h>,
<unctrl.h> and <widec.h>.

Note that addwch(), mvaddwch(), mvwaddwch(), and echowchar() may be
macros.

None of these routines can use the color attribute in chtype.
curs_addwchstr(3CURSES)

NAME  curs_addwchstr, addwchstr, addwchnstr, waddwchstr, waddwchnstr, mvaddwchstr, mvaddwchnstr, mvwaddwchstr, mvwaddwchnstr – add string of wchar_t characters (and attributes) to a curses window

SYNOPSIS  cc [flag...] file... -lcurses [library...]

#include <curses.h>

int addwchstr(chtype *wchstr);
int addwchnstr(chtype *wchstr, int n);
int waddwchstr(WINDOW *win, chtype *wchstr);
int waddwchnstr(WINDOW *win, chtype *wchstr, int n);
int mvaddwchstr(int y, int x, chtype *wchstr);
int mvaddwchnstr(int y, int x, chtype *wchstr, int n);
int mvwaddwchstr(WINDOW *win, int y, int x, chtype *wchstr);
int mvwaddwchnstr(WINDOW *win, int y, int x, chtype *wchstr, int n);

DESCRIPTION  All of these routines copy wchstr, which points to a string of wchar_t characters, directly into the window image structure starting at the current cursor 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, to the maximum number that fit on the line.

The position of the window cursor is not advanced. These routines work faster than waddnwstr(3CURSES) because they merely copy wchstr into the window image structure. On the other hand, care must be taken when using these functions because they don't perform any kind of checking (such as for the newline character), they do not advance the current cursor position, and they truncate the string, rather than wrapping it around to the new line.

RETURN VALUE  All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion, unless otherwise noted in the preceding routine descriptions.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3CURSES), waddnwstr(3CURSES), attributes(5)

NOTES  The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h> and <widec.h>.
Note that all routines except `waddwchstr()` may be macros.

None of these routines can use the color attribute in `chttype`.
NAME
curs_addwstr, addwstr, addnwstr, waddwstr, waddnwstr, mvaddwstr, mvaddnwstr, mvwaddwstr, mvwaddnwstr – add a string of wchar_t characters to a curses window and advance cursor

SYNOPSIS
cc [flag...] file... -lcurses [library...]
#include<curses.h>
int addwstr(wchar_t *wstr);
int addnwstr(wchar_t *wstr, int n);
INT WADDWSTR(WINDOW *WIN, wchar_t *wstr);
int waddnwstr(WINDOW *win, wchar_t *wstr, int n);
int mvaddwstr(int y, int x, wchar_t *wstr);
int mvaddnwstr(int y, int x, wchar_t *wstr, int n);
int mvwaddwstr(WINDOW *win, int y, int x, wchar_t *wstr);
int mvwaddnwstr(WINDOW *win, int y, int x, wchar_t *wstr, int n);

DESCRIPTION
All of these routines write all the characters of the null-terminated wchar_t character string wstr on the given window. The effect is similar to calling waddwch(3CURSES) once for each wchar_t character in the string. The four routines with n as the last argument write at most n wchar_t characters. If n is negative, then the entire string will be added.

RETURN VALUE
All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURES), waddwch(3CURSES), attributes(5)

NOTES
The header file <curses.h> automatically includes the header files <stdio.h>, <ncr1.h> and <widec.h>.

Note that all of these routines except waddwstr() and waddnwstr() may be macros.
NAME
curs_alecompat, movenextch, wmovenextch, moveprevch, wmoveprevch, adjcurspos,
wadjcurspos – these functions are added to ALE curses library for moving the cursor by character.

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int movenextch(void);
int wmovenextch(WINDOW *win);
int moveprevch(void);
int wmoveprevch(WINDOW *win);
int adjcurspos(void);
int wadjcurspos(WINDOW *win);

DESCRIPTION
movenextch() and wmovenextch() move the cursor to the next character to the right. If the next character is a multicolumn character, the cursor is positioned on the first (left-most) column of that character. The new cursor position will be on the next character, even if the cursor was originally positioned on the left-most column of a multicolumn character. Note that the simple cursor increment (++x) does not guarantee movement to the next character, if the cursor was originally positioned on a multicolumn character. getyx(3CURSES) can be used to find the new position.

moveprevc() and wmoveprevch() routines are the opposite of movenextch() and wmovenextch(), moving the cursor to the left-most column of the previous character.

adjcurspos() and wadjcurspos() move the cursor to the first(left-most) column of the multicolumn character that the cursor is presently on. If the cursor is already on the first column, or if the cursor is on a single-column character, these routines will have no effect.

RETURN VALUE
All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), getyx(3CURSES), attributes(5)

NOTES
The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h> and <widec.h>.

Note that movenextch(), moveprevch(), and adjcurspos() may be macros.
NAME
curs_attr(3CURSES)

SYNOPSIS
c Curry("flag ... ", file ...", -lcurses [ library ... ]
#include <curses.h>
int attroff(int attrs);
int wattroff(WINDOW *win, int attrs);
int attron(int attrs);
int wattron(WINDOW *win, int attrs);
int attrset(int attrs);
int wattrset(WINDOW *win, int attrs);
int standend(void);
int wstandend(WINDOW *win);
int standout(void);
int wstandout(WINDOW *win);

DESCRIPTION
All of these routines manipulate the current attributes of the named window. The
current attributes of a window are applied to all characters that are written into the
window with waddch(), waddstr(), and wprintw(). 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 are
displayed as the graphic rendition of characters put on the screen.

The routine attrset() sets the current attributes of the given window to attrs. The
routine attroff() turns off the named attributes without turning any other
attributes on or off. The routine attron() turns on the named attributes without
affecting any others. The routine standout() is the same as attron(A_STANDOUT).
The routine standend() is the same as attrset(), that is, it turns off all attributes.

Attributes
The following video attributes, defined in <curses.h>, can be passed to the routines
attron(), attroff(), and attrset(), or OR-ed with the characters passed to
addch().

A_STANDOUT Best highlighting mode of the terminal
A_UNDERLINE Underlining
A_REVERSE Reverse video
A_BLINK Blinking
A_DIM Half bright
A_BOLD Extra bright or bold
A_ALTCHARSET Alternate character set

man pages section 3: Curses Library Functions • Last Revised 31 Dec 1996
A_CHARTEXT Bit-mask to extract a character

COLOR_PAIR(n) Color-pair number n

The following macro is the reverse of COLOR_PAIR(n):

PAIR_NUMBER(attrs) Returns the pair number associated with the COLOR_PAIR(n) attribute

RETURN VALUES These routines always return 1.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curs_addch(3CURSES), curs_addstr(3CURSES), curs_printw(3CURSES), curses(3CURSES), attributes(5)

NOTES The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.

Note that attroff(), wattroff(), attron(), wattron(), wattrset(), standend(), and standout() may be macros.
NAME
curs_beep, beep, flash – curses bell and screen flash routines

SYNOPSIS
curses bell and screen flash routines

```
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int beep(void);
int flash(void);
```

DESCRIPTION
The beep() and flash() routines are used to signal the terminal user. The routine
beep() sounds the audible alarm on the terminal, if possible; if that is not possible, it
flashes the screen (visible bell), if that is possible. The routine flash() flashes the
screen, and if that is not possible, sounds the audible signal. If neither signal is
possible, nothing happens. Nearly all terminals have an audible signal (bell or beep),
but only some can flash the screen.

RETURN VALUES
These routines always return 0.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

```
ATTRIBUTE TYPE | ATTRIBUTE VALUE
----------------|------------------
MT-Level        | Unsafe
```

SEE ALSO
curses(3CURSES), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and
<unctrl.h>.
NAME
curs_bkgd, bkgd, bkgdset, wbkgdset, wbkgd – curses window background manipulation routines

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int bkgd(chtype ch);
void bkgdset(chtype ch);
void wbkgdset(WINDOW *win, chtype ch);
int wbkgd(WINDOW *win, chtype ch);

DESCRIPTION
The bkgdset() and wbkgdset() routines manipulate the background of the named window. Background is a chtype consisting of any combination of attributes and a character. The attribute part of the background is combined (ORed) with all 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.

RETURN VALUES
bkgd() and wbkgd() return the integer OK, or a non-negative integer, if immedok() is set. See curs_outopts(3CURSES).

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_addch(3CURSES), curs_outopts(3CURSES), curses(3CURSES), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <uncrtl.h>.

Note that bkgdset() and bkgd() may be macros.
With the `border()`, `wborder()`, and `box()` routines, a border is drawn around the edges of the window. The arguments and attributes are:

- `ls`: left side of the border
- `rs`: right side of the border
- `ts`: top side of the border
- `bs`: bottom side of the border
- `tl`: top left-hand corner
- `tr`: top right-hand corner
- `bl`: bottom left-hand corner
- `br`: bottom right-hand corner

If any of these arguments is zero, then the following default values (defined in `<curses.h>`) are used respectively instead: `ACS_VLINE`, `ACS_VLINE`, `ACS_HLINE`, `ACS_HLINE`, `ACS_ULCORNER`, `ACS_URCORNER`, `ACS_BLCORNER`, `ACS_BRCORNER`.

`box(win, verch, horch)` is a shorthand for the following call:

```c
wborder(win, verch, verch, horch, horch, 0, 0, 0, 0)
```

`hline()` and `whline()` draw a horizontal (left to right) line using `ch` starting at the current cursor position in the window. The current cursor position is not changed. The line is at most `n` characters long, or as many as fit into the window.
vline() and wvline() draw a vertical (top to bottom) line using \textit{ch} starting at the current cursor position in the window. The current cursor position is not changed. The line is at most \textit{n} characters long, or as many as fit into the window.

**RETURN VALUES**

All routines return the integer \textit{OK}, or a non-negative integer if \textit{immedok()} is set. See \texttt{curs_outopts(3CURSES)}.

**ATTRIBUTES**

See \texttt{attributes(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

\texttt{curs_outopts(3CURSES)}, \texttt{curses(3CURSES)}, \texttt{attributes(5)}

**NOTES**

The header \texttt{<curses.h>} automatically includes the headers \texttt{<stdio.h>} and \texttt{<unctrl.h>}.

Note that \texttt{border()} and \texttt{box()} may be macros.
NAME
curs_clear, erase, werase, clear, wclear, clrtobot, wclrtobot, clrtoeol, wclrtoeol – clear all
or part of a curses window

SYNOPSIS
cce [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int erase(void);
int werase(WINDOW *win);
int clear(void);
int wclear(WINDOW *win);
int clrtobot(void);
int wclrtobot(WINDOW *win);
int clrtoeol(void);
int wclrtoeol(WINDOW *win);

DESCRIPTION
The erase() and werase() routines copy blanks to every position in the window.
The clear() and wclear() routines are like erase() and werase(), but they also
call clearok(), so that the screen is cleared completely on the next call to
wrefresh() for that window and repainted from scratch.
The clrtobot() and wclrtobot() routines erase all lines below the cursor in the
window. Also, the current line to the right of the cursor, inclusive, is erased.
The clrtoeol() and wclrtoeol() routines erase the current line to the right of the
cursor, inclusive.

RETURN VALUES
All routines return the integer OK, or a non-negative integer if immedok() is set. See
curs_outopts(3CURSES).

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_outopts(3CURSES), curs_refresh(3CURSES), curses(3CURSES),
attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and
<unctrl.h>.
Note that erase(), werase(), clear(), wclear(), clrtobot(), and
clrtoeol() may be macros.
NAME curses_color, start_color, init_pair, init_color, has_colors, can_change_color, color_content, pair_content – curses color manipulation routines

SYNOPSIS

```c
#include <curses.h>

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 *redp, short *greenp, short *bluep);
int pair_content(short pair, short *fgp, short *bpg);
```

Overview

curses provides routines that manipulate color on color alphanumeric terminals. To use these routines start_color() must be called, usually right after initscr(). See curs_initscr(3CURSES). Colors are always used in pairs (referred to as color-pairs). A color-pair consists of a foreground color (for characters) and a background color (for the field on which the characters are displayed). A programmer initializes a color-pair with the routine init_pair. After it has been initialized, COLOR_PAIR(n), a macro defined in <curses.h>, can be used in the same ways other video attributes can be used. If a terminal is capable of redefining colors, the programmer can use the routine init_color() to change the definition of a color. The routines has_colors() and can_change_color() return TRUE or FALSE, depending on whether the terminal has color capabilities and whether the programmer can change the colors. The routine color_content() allows a programmer to identify the amounts of red, green, and blue components in an initialized color. The routine pair_content() allows a programmer to find out how a given color-pair is currently defined.

Routine Descriptions

The start_color() routine requires no arguments. It must be called if the programmer wants to use colors, and before any other color manipulation routine is called. It is good practice to call this routine right after initscr(). 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 can support). It also restores the colors on the terminal to the values they had when the terminal was just turned on.

The init_pair() routine changes the definition of a color-pair. It takes three arguments: the number of the color-pair to be changed, the foreground color number, and the background color number. The value of the first argument must be between 1 and COLOR_PAIRS-1. The value of the second and third arguments must be between 0 and COLORS. If the color-pair was previously initialized, the screen is refreshed and all occurrences of that color-pair is changed to the new definition.
The `init_color()` routine changes the definition of a color. It takes four arguments: the number of the color to be changed followed by three RGB values (for the amounts of red, green, and blue components). The value of the first argument must be between 0 and COLORS. (See the section Colors for the default color index.) Each of the last three arguments must be a value between 0 and 1000. When `init_color()` is used, all occurrences of that color on the screen immediately change to the new definition.

The `has_colors()` routine requires no arguments. It returns TRUE if the terminal can manipulate colors; otherwise, it returns FALSE. This routine facilitates writing terminal-independent programs. For example, a programmer can use it to decide whether to use color or some other video attribute.

The `can_change_color()` routine requires no arguments. It returns TRUE if the terminal supports colors and can change their definitions; other, it returns FALSE. This routine facilitates writing terminal-independent programs.

The `color_content()` routine gives users a way to find the intensity of the red, green, and blue (RGB) components in a color. It requires four arguments: the color number, and three addresses of shorts for storing the information about the amounts of red, green, and blue components in the given color. The value of the first argument must be between 0 and COLORS. The values that are stored at the addresses pointed to by the last three arguments are between 0 (no component) and 1000 (maximum amount of component).

The `pair_content()` routine allows users to find out what colors a given color-pair consists of. It requires three arguments: the color-pair number, and two addresses of shorts for storing the foreground and the background color numbers. The value of the first argument must be between 1 and COLOR_PAIRS−1. The values that are stored at the addresses pointed to by the second and third arguments are between 0 and COLORS.

Colors

In `<curses.h>` the following macros are defined. These are the default colors. `curses` also assumes that COLOR_BLACK is the default background color for all terminals.

```
COLOR_BLACK
COLOR_RED
COLOR_GREEN
COLOR_YELLOW
COLOR_CYAN
COLOR_BLUE
COLOR_MAGENTA
COLOR_WHITE
```

RETURN VALUES

All routines that return an integer return ERR upon failure and OK upon successful completion.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:
curs_color(3CURSES)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curs_attr(3CURSES), curs_initscr(3CURSES), curses(3CURSES), attributes(5)

NOTES The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.
### NAME
curscr – current window

### SYNOPSIS
```c
#include <curses.h>

extern WINDOW *curscr;
```

### DESCRIPTION
The external variable `curscr` points to an internal data structure. It can be specified as an argument to certain functions such as `clearok(3XCURSES)`.

### SEE ALSO
`clearok(3XCURSES)`
curses delch, delch, wdelch, mvdelch, mvwdelch – delete character under cursor in a curses window

SYNOPSIS

```
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int delch(void);
int wdelch(WINDOW *win);
int mvdelch(int y, int x);
int mvwdelch(WINDOW *win, int y, int x);
```

DESCRIPTION

With these routines the character under the cursor in the window is deleted; all characters to the right of the cursor 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 specified). This does not imply use of the hardware delete character feature.

RETURN VALUES

All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

curses(3CURSES), attributes(5)

NOTES

The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.

Note that delch(), mvdelch(), and mvwdelch() may be macros.
NAME  
curs_deleteln, deleteln, wdeleteln, insdelln, winsdelln, insertln, winsertln – delete and insert lines in a curses window

SYNOPSIS  
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int deleteln(void);
int wdeleteln(WINDOW *win);
int insdelln(int n);
int winsdelln(WINDOW *win, int n);
int insertln(void);
int winsertln(WINDOW *win);

DESCRIPTION  
With the deleteln() and wdeleteln() routines, 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. This does not imply use of a hardware delete line feature.

With the insdelln() and winsdelln() routines, for positive n, insert n lines into the specified window above the current line. The n bottom lines are lost. For negative n, delete n lines (starting with the one under the cursor), and move the remaining lines up. The bottom n lines are cleared. The current cursor position remains the same.

With the insertln() and insertln() routines, a blank line is inserted above the current line and the bottom line is lost. This does not imply use of a hardware insert line feature.

RETURN VALUES  
All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.

ATTRIBUTES  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curses(3CURSES), attributes(5)

NOTES  
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.

Note that all but winsdelln() may be macros.
NAME

curses – CRT screen handling and optimization package

SYNOPSIS

c c [ flag... ] file... -lcurses [ library... ]
#include <curses.h>

DESCRIPTION

The *curses* library routines give the user a terminal-independent method of updating character screens with reasonable optimization.

The *curses* package allows: overall screen, window and pad manipulation; output to windows and pads; reading terminal input; control over terminal and *curses* input and output options; environment query routines; color manipulation; use of soft label keys; terminfo access; and access to low-level *curses* routines.

To initialize the routines, the routine `initscr()` or `newterm()` must be called before any of the other routines that deal with windows and screens are used. The routine `endwin()` must be called before exiting. To get character-at-a-time input without echoing (most interactive, screen oriented programs want this), the following sequence should be used:

```
initscr, cbreak, noecho;
```

Most programs would additionally use the sequence:

```
nonl, intrflush(stdscr, FALSE), keypad(stdscr, TRUE);
```

Before a *curses* program is run, the tab stops of the terminal should be set and its initialization strings, if defined, must be output. This can be done by executing the `tput init` command after the shell environment variable `TERM` has been exported. (See `terminfo(4)` for further details.)

The *curses* library permits manipulation of data structures, called *windows*, which can be thought of as two-dimensional arrays of characters representing all or part of a CRT screen. A default window called *stdscr*, which is the size of the terminal screen, is supplied. Others may be created with `newwin(3CURSES)`.

Windows are referred to by variables declared as `WINDOW*`. These data structures are manipulated with routines described on 3X pages (whose names begin "curs_"), among which the most basic routines are `move(3CURSES)` and `addch(3CURSES)`. More general versions of these routines are included with names beginning with `w`, allowing the user to specify a window. The routines not beginning with `w` affect `stdscr`.

After using routines to manipulate a window, `refresh(3CURSES)` is called, telling *curses* to make the user’s CRT screen look like `stdscr`. The characters in a window are actually of type `chtype`, (character and attribute data) so that other information about the character may also be stored with each character.
Special windows called pads may also be manipulated. These are windows which are not constrained to the size of the screen and whose contents need not be completely displayed. See curs_pad(3CURSES) for more information.

In addition to drawing characters on the screen, video attributes and colors may be included, causing the characters to show up in such modes as underlined, in reverse video, or in color on terminals that support such display enhancements. Line drawing characters may be specified to be output. On input, curses is also able to translate arrow and function keys that transmit escape sequences into single values. The video attributes, line drawing characters, and input values use names, defined in <curses.h>, such as A_REVERSE, ACS_HLINE, and KEY_LEFT.

If the environment variables LINES and COLUMNS are set, or if the program is executing in a window environment, line and column information in the environment will override information read by terminfo. This would effect a program running in an AT&T 630 layer, for example, where the size of a screen is changeable.

If the environment variable TERMINFO is defined, any program using curses checks for a local terminal definition before checking in the standard place. For example, if TERM is set to att4424, then the compiled terminal definition is found in

```
/usr/share/lib/terminfo/a/att4424.
```

(The ‘a’ is copied from the first letter of att4424 to avoid creation of huge directories.) However, if TERMINFO is set to $HOME/myterms, curses first checks

```
$HOME/myterms/a/att4424,
```

and if that fails, it then checks

```
/usr/share/lib/terminfo/a/att4424.
```

This is useful for developing experimental definitions or when write permission in

```
/usr/share/lib/terminfo
```

is not available.

The integer variables LINES and COLS are defined in <curses.h> and will be filled in by initscr with the size of the screen. The constants TRUE and FALSE have the values 1 and 0, respectively.

The curses routines also define the WINDOW * variable curscr which is used for certain low-level operations like clearing and redrawing a screen containing garbage. The curscr can be used in only a few routines.

The number of bytes and the number of columns to hold a character from the supplementary character set is locale-specific (locale category LC_CTYPE) and can be specified in the character class table.
For editing, operating at the character level is entirely appropriate. For screen formatting, arbitrary movement of characters on screen is not desirable.

Overwriting characters (addch, for example) operates on a screen level. Overwriting a character by a character that requires a different number of columns may produce orphaned columns. These orphaned columns are filled with background characters.

Inserting characters (insch, for example) operates on a character level (that is, at the character boundaries). The specified character is inserted right before the character, regardless of which column of a character the cursor points to. Before insertion, the cursor position is adjusted to the first column of the character.

As with inserting characters, deleting characters (delch, for example) operates on a character level (that is, at the character boundaries). The character at the cursor is deleted whichever column of the character the cursor points to. Before deletion, the cursor position is adjusted to the first column of the character.

A multi-column character cannot be put on the last column of a line. When such attempts are made, the last column is set to the background character. In addition, when such an operation creates orphaned columns, the orphaned columns are filled with background characters.

Overlapping and overwriting a window follows the operation of overwriting characters around its edge. The orphaned columns, if any, are handled as in the character operations.

The cursor is allowed to be placed anywhere in a window. If the insertion or deletion is made when the cursor points to the second or later column position of a character that holds multiple columns, the cursor is adjusted to the first column of the character before the insertion or deletion.

Many curses routines have two or more versions. The routines prefixed with w require a window argument. The routines prefixed with p require a pad argument. Those without a prefix generally use stdscr.

The routines prefixed with mv require an x and y coordinate to move to before performing the appropriate action. The mv routines imply a call to move(3CURSES) before the call to the other routine. The coordinate y always refers to the row (of the window), and x always refers to the column. The upper left-hand corner is always (0,0), not (1,1).

The routines prefixed with mvw take both a window argument and x and y coordinates. The window argument is always specified before the coordinates.

In each case, win is the window affected, and pad is the pad affected; win and pad are always pointers to type WINDOW.

Option setting routines require a Boolean flag bf with the value TRUE or FALSE; bf is always of type bool. The variables ch and attrs below are always of type chtype. The
types WINDOW, SCREEN, bool, and chtype are defined in <curses.h>. The type TERMINAL is defined in <term.h>. All other arguments are integers.

The following table lists each curses routine and the name of the manual page on which it is described.

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Manual Page Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>addch</td>
<td>curs_addch(3CURSES)</td>
</tr>
<tr>
<td>addchnstr</td>
<td>curs_addchstr(3CURSES)</td>
</tr>
<tr>
<td>addchstr</td>
<td>curs_addchstr(3CURSES)</td>
</tr>
<tr>
<td>addnstr</td>
<td>curs_addstr(3CURSES)</td>
</tr>
<tr>
<td>addnwstr</td>
<td>curs_addwstr(3CURSES)</td>
</tr>
<tr>
<td>addstr</td>
<td>curs_addstr(3CURSES)</td>
</tr>
<tr>
<td>addwch</td>
<td>curs_addwch(3CURSES)</td>
</tr>
<tr>
<td>addwchnstr</td>
<td>curs_addwchstr(3CURSES)</td>
</tr>
<tr>
<td>addwchstr</td>
<td>curs_addwchstr(3CURSES)</td>
</tr>
<tr>
<td>addwstr</td>
<td>curs_addwstr(3CURSES)</td>
</tr>
<tr>
<td>adjcurspos</td>
<td>curs_alecompat(3CURSES)</td>
</tr>
<tr>
<td>attroff</td>
<td>curs_attr(3CURSES)</td>
</tr>
<tr>
<td>attron</td>
<td>curs_attr(3CURSES)</td>
</tr>
<tr>
<td>attrset</td>
<td>curs_attr(3CURSES)</td>
</tr>
<tr>
<td>baudrate</td>
<td>curs_termattrs(3CURSES)</td>
</tr>
<tr>
<td>beep</td>
<td>curs_beep(3CURSES)</td>
</tr>
<tr>
<td>bkgd</td>
<td>curs_bkgd(3CURSES)</td>
</tr>
<tr>
<td>bgdset</td>
<td>curs_bkgd(3CURSES)</td>
</tr>
<tr>
<td>border</td>
<td>curs_border(3CURSES)</td>
</tr>
<tr>
<td>box</td>
<td>curs_border(3CURSES)</td>
</tr>
<tr>
<td>can_change_color</td>
<td>curs_color(3CURSES)</td>
</tr>
<tr>
<td>cbreak</td>
<td>curs_inopts(3CURSES)</td>
</tr>
<tr>
<td>clear</td>
<td>curs_clear(3CURSES)</td>
</tr>
<tr>
<td>clearok</td>
<td>curs_outopts(3CURSES)</td>
</tr>
<tr>
<td>clrtoeol</td>
<td>curs_clear(3CURSES)</td>
</tr>
<tr>
<td>clrtoeol</td>
<td>curs_clear(3CURSES)</td>
</tr>
<tr>
<td>color_content</td>
<td>curs_color(3CURSES)</td>
</tr>
</tbody>
</table>
curses(3CURSES)

<table>
<thead>
<tr>
<th>Function</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>copywin</td>
<td>curs_overlay(3CURSES)</td>
</tr>
<tr>
<td>curs_set</td>
<td>curs_kernel(3CURSES)</td>
</tr>
<tr>
<td>def_prog_mode</td>
<td>curs_kernel(3CURSES)</td>
</tr>
<tr>
<td>def_shell_mode</td>
<td>curs_kernel(3CURSES)</td>
</tr>
<tr>
<td>del_curterm</td>
<td>curs_terminfo(3CURSES)</td>
</tr>
<tr>
<td>delay_output</td>
<td>curs_util(3CURSES)</td>
</tr>
<tr>
<td>delch</td>
<td>curs_delch(3CURSES)</td>
</tr>
<tr>
<td>deleteln</td>
<td>curs_deleteeln(3CURSES)</td>
</tr>
<tr>
<td>delscreen</td>
<td>curs_initscr(3CURSES)</td>
</tr>
<tr>
<td>delwin</td>
<td>curs_window(3CURSES)</td>
</tr>
<tr>
<td>derwin</td>
<td>curs_window(3CURSES)</td>
</tr>
<tr>
<td>dupupdate</td>
<td>curs_refresh(3CURSES)</td>
</tr>
<tr>
<td>dupwin</td>
<td>curs_window(3CURSES)</td>
</tr>
<tr>
<td>echo</td>
<td>curs_inopts(3CURSES)</td>
</tr>
<tr>
<td>echochar</td>
<td>curs_addch(3CURSES)</td>
</tr>
<tr>
<td>echowchar</td>
<td>curs_addwch(3CURSES)</td>
</tr>
<tr>
<td>endwin</td>
<td>curs_initscr(3CURSES)</td>
</tr>
<tr>
<td>erase</td>
<td>curs_clear(3CURSES)</td>
</tr>
<tr>
<td>erasechar</td>
<td>curs_termattrs(3CURSES)</td>
</tr>
<tr>
<td>filter</td>
<td>curs_util(3CURSES)</td>
</tr>
<tr>
<td>flash</td>
<td>curs_beep(3CURSES)</td>
</tr>
<tr>
<td>flushinp</td>
<td>curs_util(3CURSES)</td>
</tr>
<tr>
<td>getbegyx</td>
<td>curs_getyx(3CURSES)</td>
</tr>
<tr>
<td>getch</td>
<td>curs_getch(3CURSES)</td>
</tr>
<tr>
<td>getmaxyx</td>
<td>curs_getyx(3CURSES)</td>
</tr>
<tr>
<td>getnwstr</td>
<td>curs_getwstr(3CURSES)</td>
</tr>
<tr>
<td>getparyx</td>
<td>curs_getyx(3CURSES)</td>
</tr>
<tr>
<td>getstr</td>
<td>curs_getstr(3CURSES)</td>
</tr>
<tr>
<td>getsyx</td>
<td>curs_kernel(3CURSES)</td>
</tr>
<tr>
<td>getwch</td>
<td>curs_getwch(3CURSES)</td>
</tr>
<tr>
<td>getwin</td>
<td>curs_util(3CURSES)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>getwstr</td>
<td>Get a wide string from the terminal</td>
</tr>
<tr>
<td>getyx</td>
<td>Get a y coordinate of the cursor</td>
</tr>
<tr>
<td>halfdelay</td>
<td>Set the half delay of the terminal</td>
</tr>
<tr>
<td>has_colors</td>
<td>Check if the terminal has colors</td>
</tr>
<tr>
<td>has_ic</td>
<td>Check if the terminal has inverse characteristics</td>
</tr>
<tr>
<td>has_Il</td>
<td>Check if the terminal has intensify</td>
</tr>
<tr>
<td>idcok</td>
<td>Set the idcok option</td>
</tr>
<tr>
<td>idlok</td>
<td>Set the idlok option</td>
</tr>
<tr>
<td>immedok</td>
<td>Set the immedok option</td>
</tr>
<tr>
<td>inch</td>
<td>Insert a character at the current cursor position</td>
</tr>
<tr>
<td>inchstr</td>
<td>Insert a string at the current cursor position</td>
</tr>
<tr>
<td>init_color</td>
<td>Initialize the color</td>
</tr>
<tr>
<td>init_pair</td>
<td>Initialize the pair</td>
</tr>
<tr>
<td>initscr</td>
<td>Initialize a new terminal</td>
</tr>
<tr>
<td>innstr</td>
<td>Insert a character at the current cursor position</td>
</tr>
<tr>
<td>innwstr</td>
<td>Insert a wide string at the current cursor position</td>
</tr>
<tr>
<td>insch</td>
<td>Insert a character at the current cursor position</td>
</tr>
<tr>
<td>insdelln</td>
<td>Insert a delete line at the current cursor position</td>
</tr>
<tr>
<td>insertln</td>
<td>Insert a line at the current cursor position</td>
</tr>
<tr>
<td>insnst</td>
<td>Insert a string at the current cursor position</td>
</tr>
<tr>
<td>insnwstr</td>
<td>Insert a wide string at the current cursor position</td>
</tr>
<tr>
<td>insnstr</td>
<td>Insert a string at the current cursor position</td>
</tr>
<tr>
<td>instr</td>
<td>Insert a character at the current cursor position</td>
</tr>
<tr>
<td>inswch</td>
<td>Insert a character at the current cursor position</td>
</tr>
<tr>
<td>inswstr</td>
<td>Insert a wide string at the current cursor position</td>
</tr>
<tr>
<td>intrflush</td>
<td>Insert a flush line</td>
</tr>
<tr>
<td>inwch</td>
<td>Insert a character at the current cursor position</td>
</tr>
<tr>
<td>inwchnstr</td>
<td>Insert a wide char at the current cursor position</td>
</tr>
<tr>
<td>inwchstr</td>
<td>Insert a wide string at the current cursor position</td>
</tr>
<tr>
<td>inwstr</td>
<td>Insert a wide character at the current cursor position</td>
</tr>
<tr>
<td>Function</td>
<td>Library Function</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>is_linetouched</td>
<td>curs_touch(3CURSES)</td>
</tr>
<tr>
<td>is_wintouched</td>
<td>curs_touch(3CURSES)</td>
</tr>
<tr>
<td>isendwin</td>
<td>curs_initscr(3CURSES)</td>
</tr>
<tr>
<td>keyname</td>
<td>curs_util(3CURSES)</td>
</tr>
<tr>
<td>keypad</td>
<td>curs_inopts(3CURSES)</td>
</tr>
<tr>
<td>killchar</td>
<td>curs_termattrs(3CURSES)</td>
</tr>
<tr>
<td>leaveok</td>
<td>curs_outopts(3CURSES)</td>
</tr>
<tr>
<td>longname</td>
<td>curs_termattrs(3CURSES)</td>
</tr>
<tr>
<td>meta</td>
<td>curs_inopts(3CURSES)</td>
</tr>
<tr>
<td>move</td>
<td>curs_move(3CURSES)</td>
</tr>
<tr>
<td>movenextch</td>
<td>curs_alecompat(3CURSES)</td>
</tr>
<tr>
<td>moveprevch</td>
<td>curs_alecompat(3CURSES)</td>
</tr>
<tr>
<td>mvaddch</td>
<td>curs_addch(3CURSES)</td>
</tr>
<tr>
<td>mvaddchnstr</td>
<td>curs_addchstr(3CURSES)</td>
</tr>
<tr>
<td>mvaddchstr</td>
<td>curs_addchstr(3CURSES)</td>
</tr>
<tr>
<td>mvaddnstr</td>
<td>curs_addstr(3CURSES)</td>
</tr>
<tr>
<td>mvaddnwstr</td>
<td>curs_addwstr(3CURSES)</td>
</tr>
<tr>
<td>mvaddstr</td>
<td>curs_addstr(3CURSES)</td>
</tr>
<tr>
<td>mvaddwch</td>
<td>curs_addwch(3CURSES)</td>
</tr>
<tr>
<td>mvaddwchnstr</td>
<td>curs_addwchstr(3CURSES)</td>
</tr>
<tr>
<td>mvaddwchstr</td>
<td>curs_addwchstr(3CURSES)</td>
</tr>
<tr>
<td>mvaddwstr</td>
<td>curs_addwstr(3CURSES)</td>
</tr>
<tr>
<td>mvcur</td>
<td>curs_terminfo(3CURSES)</td>
</tr>
<tr>
<td>mvdelch</td>
<td>curs_delch(3CURSES)</td>
</tr>
<tr>
<td>mvderwin</td>
<td>curs_window(3CURSES)</td>
</tr>
<tr>
<td>mvgetch</td>
<td>curs_getch(3CURSES)</td>
</tr>
<tr>
<td>mvgetnwstr</td>
<td>curs_getwstr(3CURSES)</td>
</tr>
<tr>
<td>mvgetstr</td>
<td>curs_getstr(3CURSES)</td>
</tr>
<tr>
<td>mvgetwch</td>
<td>curs_getwch(3CURSES)</td>
</tr>
<tr>
<td>mvgetwstr</td>
<td>curs_getwstr(3CURSES)</td>
</tr>
<tr>
<td>mvinch</td>
<td>curs_inch(3CURSES)</td>
</tr>
</tbody>
</table>
**curses(3CURSES)**

<table>
<thead>
<tr>
<th>Function</th>
<th>Equivalent Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvchnstr</td>
<td>curs_inchstr(3CURSES)</td>
</tr>
<tr>
<td>mvchstr</td>
<td>curs_inchstr(3CURSES)</td>
</tr>
<tr>
<td>mvinnstr</td>
<td>curs_instr(3CURSES)</td>
</tr>
<tr>
<td>mvinnwstr</td>
<td>curs_inwstr(3CURSES)</td>
</tr>
<tr>
<td>mvinsch</td>
<td>curs_insch(3CURSES)</td>
</tr>
<tr>
<td>mvinsnstr</td>
<td>curs_insstr(3CURSES)</td>
</tr>
<tr>
<td>mvinsnwstr</td>
<td>curs_inswwstr(3CURSES)</td>
</tr>
<tr>
<td>mvinsstr</td>
<td>curs_insstr(3CURSES)</td>
</tr>
<tr>
<td>mvinstr</td>
<td>curs_instr(3CURSES)</td>
</tr>
<tr>
<td>mvinswch</td>
<td>curs_inswwch(3CURSES)</td>
</tr>
<tr>
<td>mvinswstr</td>
<td>curs_inswwstr(3CURSES)</td>
</tr>
<tr>
<td>mvinwch</td>
<td>curs_inwch(3CURSES)</td>
</tr>
<tr>
<td>mvinwchnstr</td>
<td>curs_inwchstr(3CURSES)</td>
</tr>
<tr>
<td>mvinwchstr</td>
<td>curs_inwchstr(3CURSES)</td>
</tr>
<tr>
<td>mvinwstr</td>
<td>curs_inwstr(3CURSES)</td>
</tr>
<tr>
<td>mvprintw</td>
<td>curs_printw(3CURSES)</td>
</tr>
<tr>
<td>mvscanw</td>
<td>curs_scanw(3CURSES)</td>
</tr>
<tr>
<td>mvwaddch</td>
<td>curs_addch(3CURSES)</td>
</tr>
<tr>
<td>mvwaddchnstr</td>
<td>curs_addchstr(3CURSES)</td>
</tr>
<tr>
<td>mvwaddchstr</td>
<td>curs_addchstr(3CURSES)</td>
</tr>
<tr>
<td>mvwaddnstr</td>
<td>curs_addstr(3CURSES)</td>
</tr>
<tr>
<td>mvwaddnwstr</td>
<td>curs_addnwstr(3CURSES)</td>
</tr>
<tr>
<td>mvwaddstr</td>
<td>curs_addstr(3CURSES)</td>
</tr>
<tr>
<td>mvwaddwch</td>
<td>curs_addwch(3CURSES)</td>
</tr>
<tr>
<td>mvwaddwchnstr</td>
<td>curs_addwchstr(3CURSES)</td>
</tr>
<tr>
<td>mvwaddwchstr</td>
<td>curs_addwchstr(3CURSES)</td>
</tr>
<tr>
<td>mvwaddwstr</td>
<td>curs_addwstr(3CURSES)</td>
</tr>
<tr>
<td>mvwdelch</td>
<td>curs_delch(3CURSES)</td>
</tr>
<tr>
<td>mvwgetch</td>
<td>curs_getch(3CURSES)</td>
</tr>
<tr>
<td>mvwgetnstr</td>
<td>curs_getnstr(3CURSES)</td>
</tr>
<tr>
<td>mvwgetstr</td>
<td>curs_getstr(3CURSES)</td>
</tr>
<tr>
<td>mvwgetwch</td>
<td>curs_getwch(3CURSES)</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>mvwgetwstr</td>
<td>curs_getwstr(3CURSES)</td>
</tr>
<tr>
<td>mvwin</td>
<td>curs_window(3CURSES)</td>
</tr>
<tr>
<td>mvwinch</td>
<td>curs_inch(3CURSES)</td>
</tr>
<tr>
<td>mvwinchnstr</td>
<td>curs_inchstr(3CURSES)</td>
</tr>
<tr>
<td>mvwinchstr</td>
<td>curs_inchstr(3CURSES)</td>
</tr>
<tr>
<td>mvwinnstr</td>
<td>curs_instr(3CURSES)</td>
</tr>
<tr>
<td>mvwinnwstr</td>
<td>curs_inwstr(3CURSES)</td>
</tr>
<tr>
<td>mvwinsch</td>
<td>curs_insch(3CURSES)</td>
</tr>
<tr>
<td>mvwinsnstr</td>
<td>curs_insstr(3CURSES)</td>
</tr>
<tr>
<td>mvwinsstr</td>
<td>curs_insstr(3CURSES)</td>
</tr>
<tr>
<td>mvwinstr</td>
<td>curs_instr(3CURSES)</td>
</tr>
<tr>
<td>mvwinswch</td>
<td>curs_inswch(3CURSES)</td>
</tr>
<tr>
<td>mvwinswstr</td>
<td>curs_inswstr(3CURSES)</td>
</tr>
<tr>
<td>mvwinwch</td>
<td>curs_inwch(3CURSES)</td>
</tr>
<tr>
<td>mvwinwchnstr</td>
<td>curs_inwchstr(3CURSES)</td>
</tr>
<tr>
<td>mvwinwchstr</td>
<td>curs_inwchstr(3CURSES)</td>
</tr>
<tr>
<td>mvwinwstr</td>
<td>curs_inwstr(3CURSES)</td>
</tr>
<tr>
<td>mvwprintw</td>
<td>curs_printw(3CURSES)</td>
</tr>
<tr>
<td>mvwscanw</td>
<td>curs_scanw(3CURSES)</td>
</tr>
<tr>
<td>napms</td>
<td>curs_kernel(3CURSES)</td>
</tr>
<tr>
<td>newpad</td>
<td>curs_pad(3CURSES)</td>
</tr>
<tr>
<td>newterm</td>
<td>curs_initscr(3CURSES)</td>
</tr>
<tr>
<td>newwin</td>
<td>curs_window(3CURSES)</td>
</tr>
<tr>
<td>nl</td>
<td>curs_outopts(3CURSES)</td>
</tr>
<tr>
<td>nocbreak</td>
<td>curs_inopts(3CURSES)</td>
</tr>
<tr>
<td>nodelay</td>
<td>curs_inopts(3CURSES)</td>
</tr>
<tr>
<td>noecho</td>
<td>curs_inopts(3CURSES)</td>
</tr>
<tr>
<td>nonl</td>
<td>curs_outopts(3CURSES)</td>
</tr>
<tr>
<td>noqiflush</td>
<td>curs_inopts(3CURSES)</td>
</tr>
<tr>
<td>noraw</td>
<td>curs_inopts(3CURSES)</td>
</tr>
</tbody>
</table>
curses(3CURSES)

notimeout (curs_inopts(3CURSES))
overlay (curs_overlay(3CURSES))
overwrite (curs_overlay(3CURSES))
pair_content (curs_color(3CURSES))
pechochar (curs_pad(3CURSES))
pechowchar (curs_pad(3CURSES))
pnoutrefresh (curs_pad(3CURSES))
prefresh (curs_pad(3CURSES))
printw (curs_printw(3CURSES))
putp (curs_terminfo(3CURSES))
putwin (curs_util(3CURSES))
qiflush (curs_inopts(3CURSES))
raw (curs_inopts(3CURSES))
redrawwin (curs_refresh(3CURSES))
refresh (curs_refresh(3CURSES))
reset_prog_mode (curs_kernel(3CURSES))
reset_shell_mode (curs_kernel(3CURSES))
resetty (curs_kernel(3CURSES))
restartterm (curs_terminfo(3CURSES))
ripoffline (curs_kernel(3CURSES))
savetty (curs_kernel(3CURSES))
scanw (curs_scanw(3CURSES))
scr_dump (curs_scr_dump(3CURSES))
scr_init (curs_scr_dump(3CURSES))
scr_restore (curs_scr_dump(3CURSES))
scr_set (curs_scr_dump(3CURSES))
scroll (curs_scroll(3CURSES))
scrollok (curs_outopts(3CURSES))
set_curterm (curs_terminfo(3CURSES))
set_term (curs_initscr(3CURSES))
setscrreg (curs_outopts(3CURSES))
<table>
<thead>
<tr>
<th>Function</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>setsyx</td>
<td>curs_kernel(3CURSES)</td>
</tr>
<tr>
<td>setterm</td>
<td>curs_terminfo(3CURSES)</td>
</tr>
<tr>
<td>setupterm</td>
<td>curs_terminfo(3CURSES)</td>
</tr>
<tr>
<td>slk_atroff</td>
<td>curs_slk(3CURSES)</td>
</tr>
<tr>
<td>slk_attron</td>
<td>curs_slk(3CURSES)</td>
</tr>
<tr>
<td>slk_atrset</td>
<td>curs_slk(3CURSES)</td>
</tr>
<tr>
<td>slk_clear</td>
<td>curs_slk(3CURSES)</td>
</tr>
<tr>
<td>slk_init</td>
<td>curs_slk(3CURSES)</td>
</tr>
<tr>
<td>slk_label</td>
<td>curs_slk(3CURSES)</td>
</tr>
<tr>
<td>slk_noutrfresh</td>
<td>curs_slk(3CURSES)</td>
</tr>
<tr>
<td>slk_refresh</td>
<td>curs_slk(3CURSES)</td>
</tr>
<tr>
<td>slk_restore</td>
<td>curs_slk(3CURSES)</td>
</tr>
<tr>
<td>slk_set</td>
<td>curs_slk(3CURSES)</td>
</tr>
<tr>
<td>slk_touch</td>
<td>curs_slk(3CURSES)</td>
</tr>
<tr>
<td>srcl</td>
<td>curs_scroll(3CURSES)</td>
</tr>
<tr>
<td>standend</td>
<td>curs_attr(3CURSES)</td>
</tr>
<tr>
<td>standout</td>
<td>curs_attr(3CURSES)</td>
</tr>
<tr>
<td>start_color</td>
<td>curs_color(3CURSES)</td>
</tr>
<tr>
<td>subpad</td>
<td>curs_pad(3CURSES)</td>
</tr>
<tr>
<td>subwin</td>
<td>curs_window(3CURSES)</td>
</tr>
<tr>
<td>syncok</td>
<td>curs_window(3CURSES)</td>
</tr>
<tr>
<td>termattrs</td>
<td>curs_termattrs(3CURSES)</td>
</tr>
<tr>
<td>termname</td>
<td>curs_termattrs(3CURSES)</td>
</tr>
<tr>
<td>tgetent</td>
<td>curs_termcap(3CURSES)</td>
</tr>
<tr>
<td>tgetflag</td>
<td>curs_termcap(3CURSES)</td>
</tr>
<tr>
<td>tgetnum</td>
<td>curs_termcap(3CURSES)</td>
</tr>
<tr>
<td>tgetstr</td>
<td>curs_termcap(3CURSES)</td>
</tr>
<tr>
<td>goto</td>
<td>curs_termcap(3CURSES)</td>
</tr>
<tr>
<td>tigetflag</td>
<td>curs_terminfo(3CURSES)</td>
</tr>
<tr>
<td>tigetnum</td>
<td>curs_terminfo(3CURSES)</td>
</tr>
<tr>
<td>tigetstr</td>
<td>curs_terminfo(3CURSES)</td>
</tr>
</tbody>
</table>

**Curses Library Functions**
timeout          curs_inopts(3CURSES)
touchline       curs_touch(3CURSES)
touchwin        curs_touch(3CURSES)
tparm           curs_terminfo(3CURSES)
tputs           curs_terminfo(3CURSES)
typeahead       curs_inopts(3CURSES)
unctrl          curs_util(3CURSES)
ungetch         curs_getch(3CURSES)
ungetwch        curs_getwch(3CURSES)
untouchwin      curs_touch(3CURSES)
use_env         curs_util(3CURSES)
vidattr         curs_terminfo(3CURSES)
vidputs         curs_terminfo(3CURSES)
vwprintw         curs_printw(3CURSES)
vwscanw         curs_scanw(3CURSES)
waddch          curs_addch(3CURSES)
waddchnstr      curs_addchstr(3CURSES)
waddchstr       curs_addchstr(3CURSES)
waddnstr        curs_addstr(3CURSES)
waddnwstr       curs_addwstr(3CURSES)
waddstr         curs_addstr(3CURSES)
waddwch         curs_addwch(3CURSES)
waddwchnstr     curs_addwchstr(3CURSES)
waddwchstr      curs_addwchstr(3CURSES)
waddwstr        curs_addwstr(3CURSES)
wadjcurspos     curs_ailecompat(3CURSES)
wattroff        curs_attr(3CURSES)
wattron         curs_attr(3CURSES)
wattrset        curs_attr(3CURSES)
wbgkd           curs_bkgd(3CURSES)
wbgkdset        curs_bkgd(3CURSES)
wborder   curs_border(3CURSES)
wclear    curs_clear(3CURSES)
wclrtobot curs_clear(3CURSES)
wclrtoeol curs_clear(3CURSES)
w cursyncup curs_window(3CURSES)
wdelch    curs_delch(3CURSES)
wdelte ln curs_deletein(3CURSES)
wechochar curs_addch(3CURSES)
wechowchar curs_addwch(3CURSES)
werase    curs_clear(3CURSES)
wgetch    curs_getch(3CURSES)
wgetnstr  curs_getstr(3CURSES)
wgetnwstr curs_getwstr(3CURSES)
wgetstr   curs_getstr(3CURSES)
wgetwch   curs_getwch(3CURSES)
wgetwstr  curs_getwstr(3CURSES)
whline    curs_border(3CURSES)
winch     curs_inch(3CURSES)
winchnstr curs_inchstr(3CURSES)
winchstr  curs_inchstr(3CURSES)
winnstr   curs_inst r(3CURSES)
winnwstr  curs_inwstr(3CURSES)
winsch    curs_insch(3CURSES)
winsdelln curs_deletein(3CURSES)
winsetln  curs_deletein(3CURSES)
winsnstr  curs_insstr(3CURSES)
winsnwstr curs_inswstr(3CURSES)
winst r   curs_insstr(3CURSES)
winst r   curs_insstr(3CURSES)
winswch   curs_inswch(3CURSES)
winswstr  curs_inswstr(3CURSES)
## RETURN VALUES
Routines that return an integer return `ERR` upon failure and an integer value other than `ERR` upon successful completion, unless otherwise noted in the routine descriptions.

All macros return the value of the `w` version, except `setscrreg()`, `wssetscrreg()`, `getyx()`, `getbegyx()`, and `getmaxyx()`. The return values of `setscrreg()`, `wssetscrreg()`, `getyx()`, `getbegyx()`, and `getmaxyx()` are undefined (that is, these should not be used as the right-hand side of assignment statements).

Routines that return pointers return `NULL` on error.

## ATTRIBUTES
See `attributes(5)` for descriptions of the following attributes:
### SEE ALSO
- `terminfo(4)`, `attributes(5)` and 3X pages whose names begin with "curs_" for detailed routine descriptions.

### NOTES
The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.
NAME
curses – introduction and overview of X/Open Curses

DESCRIPTION
The Curses screen management package conforms fully with Issue 4, Version 2 of the X/Open Curses specification. It provides a set of internationalized functions and macros for creating and modifying input and output to a terminal screen. This includes functions for creating windows, highlighting text, writing to the screen, reading from user input, and moving the cursor.

X/Open Curses is a terminal-independent package, providing a common user interface to a variety of terminal types. Its portability is facilitated by the Terminfo database which contains a compiled definition of each terminal type. By referring to the database information X/Open Curses gains access to low-level details about individual terminals.

X/Open Curses tailors its activities to the terminal type specified by the TERM environment variable. The TERM environment variable may be set in the Korn Shell (see ksh(1)) by typing:

```
export TERM=terminal_name
```

To set environment variables using other command line interfaces or shells, see the environ(5) manual page.

Three additional environment variables are useful, and can be set in the Korn Shell:

1. If you have an alternate Terminfo database containing terminal types that are not available in the system default database /usr/share/lib/terminfo, you can specify the TERMINFO environment variable to point to this alternate database:

```
export TERMINFO=path
```

   This path specifies the location of the alternate compiled Terminfo database whose structure consists of directory names 0 to 9 and a to z (which represent the first letter of the compiled terminal definition file name). The alternate database specified by TERMINFO is examined before the system default database. If the terminal type specified by TERM cannot be found in either database, the default terminal type dumb is assumed.

2. To specify a window width smaller than your screen width (for example, in situations where your communications line is slow), set the COLUMNS environment variable to the number of vertical columns you want between the left and right margins:

```
export COLUMNS=number
```

   The number of columns may be set to a number smaller than the screen size; however, if set larger than the screen or window width, the results are undefined. The value set using this environment variable takes precedence over the value normally used for the terminal.

3. To specify a window height smaller than your current screen height (for example, in situations where your communications line is slow), override the LINES environment variable by setting it to a smaller number of horizontal lines:
export LINES=number
The number of lines may be set to a number smaller than the screen height;
however, if set larger than the screen or window height, the results are undefined.
The value set using this environment variable takes precedence over the value
normally used for the terminal.

Data Types
X/Open Curses defines the following data types:

| attr_t | An integral type that holds an OR-ed set of attributes. The attributes acceptable are those which begin with the WA_prefix. |
| bool   | Boolean data type. |
| cchar_t| A type that refers to a string consisting of a spacing wide character, up to 5 non-spacing wide characters, and zero or more attributes of any type. See Attributes, Color Pairs, and Renditions. A null cchar_t object terminates arrays of cchar_t objects. |
| chtype | An integral type whose values are formed by OR-ing an "unsigned char" with a color pair and with zero or more attributes. The attributes acceptable are those which begin with the A_prefix and COLOR_PAIR(3XCURSES) |

SCREEN | An opaque data type associated with a terminal's display screen. |
TERMINAL | An opaque data type associated with a terminal. It contains information about the terminal's capabilities (as defined by terminfo), the terminal modes, and current state of input/output operations. |
wchar_t | An integral data type whose values represent wide characters. |
WINDOW | An opaque data type associated with a window. |

The X/Open Curses manual pages refer at various points to screens, windows (also subwindows, derived windows, and pads), and terminals. The following list defines each of these terms.

Screen | A screen is a terminal's physical output device. The SCREEN data type is associated with a terminal. |
Window | Window objects are two-dimensional arrays of characters and their renditions. X/Open Curses provides stdscr, a default window which is the size of of the terminal screen. You can use the newwin(3XCURSES) function to create others. |

To refer to a window, use a variable declared as WINDOW *. X/Open Curses includes both functions that modify stdscr, and more general versions that let you specify a window. |

There are three sub-types of windows:
Subwindow

A window which has been created within another window (the parent window) and whose position has been specified with absolute screen coordinates. The `derwin(3XCURSES)` and `subwin(3XCURSES)` functions can be used to create subwindows.

Derived Window

A subwindow whose position is defined relative to the parent window's coordinates rather than in absolute terms.

Pad

A special type of window that can be larger than the screen. For more information, see the `newpad(3XCURSES)` man page.

Terminal

A terminal is the input and output device which character-based applications use to interact with the user. The `TERMINAL` data type is associated with such a device.

A character's rendition consists of its attributes (such as underlining or reverse video) and its color pair (the foreground and background colors). When using `waddstr(3XCURSES)`, `waddchstr(3XCURSES)`, `wprientw(3XCURSES)`, `winsch(3XCURSES)`, and so on, the window's rendition is combined with that character's renditions. The window rendition is the attributes and color set using the `attroff(3XCURSES)` and `attr_off(3XCURSES)` sets of functions. The window's background character and rendition are set with the `bkgdset(3XCURSES)` and `bkgrndset(3XCURSES)` sets of functions.

When spaces are written to the screen, the background character and window rendition replace the space. For example, if the background rendition and character is `A_UNDERLINE|'*'`, text written to the window appears underlined and the spaces appear as underlined asterisks.

Each character written retains the rendition that it has obtained. This allows the character to be copied "as is" to or from a window with the `addchstr(3XCURSES)` or `inch(3XCURSES)` functions.

**A_ Constant Values for Attributes**

You can specify Attributes, Color Pairs, and Renditions attributes using the constants listed in the tables below. The following constants modify objects of type `chtype`:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_ALTCHARSET</td>
<td>Alternate character set</td>
</tr>
<tr>
<td>A_ATTRIBUTES</td>
<td>Bit-mask to extract attributes</td>
</tr>
<tr>
<td>A_BLINK</td>
<td>Blinking</td>
</tr>
</tbody>
</table>
### Constant Values for Attributes

The following constants modify objects of type `attr_t`:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA_ALTCHARSET</td>
<td>Alternate character set</td>
</tr>
<tr>
<td>WA_ATTRIBUTES</td>
<td>Attribute mask</td>
</tr>
<tr>
<td>WA_BLINK</td>
<td>Blinking</td>
</tr>
<tr>
<td>WA_BOLD</td>
<td>Bold</td>
</tr>
<tr>
<td>WA_DIM</td>
<td>Half-bright</td>
</tr>
<tr>
<td>WA_HORIZONTAL</td>
<td>Horizontal highlight</td>
</tr>
<tr>
<td>WA_INVIS</td>
<td>Invisible</td>
</tr>
<tr>
<td>WA_LEFT</td>
<td>Left highlist</td>
</tr>
<tr>
<td>WA_LOW</td>
<td>Low highlist</td>
</tr>
<tr>
<td>WA_PROTECT</td>
<td>Protected</td>
</tr>
<tr>
<td>WA_REVERSE</td>
<td>Reverse video</td>
</tr>
<tr>
<td>WA_RIGHT</td>
<td>Right highlight</td>
</tr>
<tr>
<td>WA_STANDOUT</td>
<td>Highlights specific to terminal</td>
</tr>
<tr>
<td>WA_TOP</td>
<td>Top highlight</td>
</tr>
<tr>
<td>WA_UNDERLINE</td>
<td>Underline</td>
</tr>
</tbody>
</table>
### Constant Description

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA_VERTICAL</td>
<td>Vertical highlight</td>
</tr>
</tbody>
</table>

### Color Macros

Colors always appear in pairs; the foreground color of the character itself and the background color of the field on which it is displayed. The following color macros are defined:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR_BLACK</td>
<td>Black</td>
</tr>
<tr>
<td>COLOR_BLUE</td>
<td>Blue</td>
</tr>
<tr>
<td>COLOR_GREEN</td>
<td>Green</td>
</tr>
<tr>
<td>COLOR_CYAN</td>
<td>Cyan</td>
</tr>
<tr>
<td>COLOR_RED</td>
<td>Red</td>
</tr>
<tr>
<td>COLOR_MAGENTA</td>
<td>Magenta</td>
</tr>
<tr>
<td>COLOR_YELLOW</td>
<td>Yellow</td>
</tr>
<tr>
<td>COLOR_WHITE</td>
<td>White</td>
</tr>
</tbody>
</table>

Together, a character’s attributes and its color pair form the character’s rendition. A character’s rendition moves with the character during any scrolling or insert/delete operations. If your terminal lacks support for the specified rendition, X/Open Curses may substitute a different rendition.

The `COLOR_PAIR(3XCURSES)` function modifies a `ccht_type` object. The `PAIR_NUMBER(3XCURSES)` function extracts the color pair from a `ccht_type` object.

### Functions for Modifying a Window’s Color

The following functions modify a window’s color:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr_set(), wattr_set()</td>
<td>Change the window’s rendition.</td>
</tr>
<tr>
<td>color_set(), wcolor_set()</td>
<td>Set the window’s color</td>
</tr>
</tbody>
</table>

### Non-Spacing Characters

When the `wcwidth(3C)` function returns a width of zero for a character, that character is called a non-spacing character. Non-spacing characters can be written to a window. Each non-spacing character is associated with a spacing character (that is, one which
Complex Characters

A non-spacing character does not have a width of zero and modifies that character. You cannot address a non-spacing character directly. Whenever you perform an X/Open Curses operation on the associated character, you are implicitly addressing the non-spacing character.

Non-spacing characters do not have a rendition. For functions that use wide characters and a rendition, X/Open Curses ignores any rendition specified for non-spacing characters. Multi-column characters have one rendition that applies to all columns spanned.

The cchar_t data type represents a complex character. A complex character may contain a spacing character, its associated non-spacing characters, and its rendition. This implementation of complex characters supports up to 5 non-spacing characters for each spacing character.

When a cchar_t object representing a non-spacing complex character is written to the screen, its rendition is not used, but rather it becomes associated with the rendition of the existing character at that location. The setcchar(3XCURSES) function initializes an object of type cchar_t. The getcchar(3XCURSES) function extracts the contents of a cchar_t object.

Display Operations

In adding internationalization support to X/Open Curses, every attempt was made to minimize the number of changes to the historical CURSES package. This enables programs written to use the historical implementation of CURSES to use the internationalized version with little or no modification. The following rules apply to the internationalized X/Open Curses package:

- The cursor can be placed anywhere in the window. Window and screen origins are (0,0).

- A multi-column character cannot be displayed in the last column, because the character would appear truncated. Instead, the background character is displayed in the last column and the multi-column character appears at the beginning of the next line. This is called wrapping.

  If the original line is the last line in the scroll region and scrolling is enabled, X/Open Curses moves the contents of each line in the region to the previous line. The first line of the region is lost. The last line of the scrolling region contains any wrapped characters. The remainder of that line is filled with the background character. If scrolling is disabled, X/Open Curses truncates any character that would extend past the last column of the screen.

- Overwrites operate on screen columns. If displaying a single-column or multi-column character results in overwriting only a portion of a multi-column character or characters, background characters are displayed in place of the non-overwritten portions.

- Insertions and deletions operate on whole characters. The cursor is moved to the first column of the character prior to performing the operation.

When windows overlap, it may be necessary to overwrite only part of a multi-column character. As mentioned earlier, the non-overwritten portions are replaced with the...
background character. This results in issues concerning the overwrite(3XCURSES), overlay(3XCURSES), copywin(3XCURSES), wnoutrefresh(3XCURSES), and wrefresh(3XCURSES) functions.

Special Characters

Some functions assign special meanings to certain special characters:

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backspace</td>
<td>Moves the cursor one column towards the beginning of the line. If the cursor was already at the beginning of the line, it remains there. All subsequent characters are added or inserted at this point.</td>
</tr>
<tr>
<td>Carriage Return</td>
<td>Moves the cursor to the beginning of the current line. If the cursor was already at the beginning of the line, it remains there. All subsequent characters are added or inserted at this point.</td>
</tr>
<tr>
<td>Newline</td>
<td>When adding characters, X/Open Curses fills the remainder of the line with the background character (effectively truncating the newline) and scrolls the window as described earlier. All subsequent characters are inserted at the start of the new line. When inserting characters, X/Open Curses fills the remainder of the line with the background character (effectively truncating the line), moves the cursor to the beginning of a new line, and scrolls the window as described earlier. All subsequent characters are placed at the start of the new line.</td>
</tr>
<tr>
<td>Tab</td>
<td>Moves subsequent characters to next horizontal tab stop. Default tab stops are set at 0, 8, 16, and so on. When adding or inserting characters, X/Open Curses inserts or adds the background character into each column until the next tab stop is reached. If there are no remaining tab stops on the current line, wrapping and scrolling occur as described earlier.</td>
</tr>
<tr>
<td>Control Characters</td>
<td>When X/Open Curses functions perform special character processing, they convert control characters to the ^X notation, where X is a single-column character (uppercase, if it is a letter) and writes that notation to the window. Functions that retrieve text from the window will retrieve the converted notation not the original.</td>
</tr>
</tbody>
</table>

X/Open Curses displays non-printable bytes, that have their high bit set, using the M–X meta notation where X is the non-printable byte with its high bit turned off.

Input Processing

There are four input modes possible with X/Open Curses that affect the behavior of input functions like getch(3XCURSES) and getnstr(3XCURSES).
In line input mode, the terminal driver handles the input of line units as well as SIGERASE and SIGKILL character processing. See termio(7I) for more information.

In this mode, the getch() and getnstr() functions will not return until a complete line has been read by the terminal driver, at which point only the requested number of bytes/characters are returned. The rest of the line unit remains unread until subsequent call to the getch() or getnstr() functions.

The functions nocbreak(3XCURSES) and noraw(3XCURSES) are used to enter this mode. These functions are described on the cbreak(3XCURSES) man page which also details which termios flags are enabled.

Of the modes available, this one gives applications the least amount of control over input. However, it is the only input mode possible on a block mode terminal.

cbreak Mode

Byte/character input provides a finer degree of control. The terminal driver passes each byte read to the application without interpreting erase and kill characters. It is the application’s responsibility to handle line editing. It is unknown whether the signal characters (SIGINTR, SIGQUIT, SIGSUSP) and flow control characters (SIGSTART, SIGSTOP) are enabled. To ensure that they are, call the noraw() function first, then call the cbreak() function.

halfdelay Mode

This is the same as the cbreak() mode with a timeout. The terminal driver waits for a byte to be received or for a timer to expire, in which case the getch() function either returns a byte or ERR respectively. This mode overrides timeouts set for an individual window with the wtimeout() function.

raw Mode

This mode provides byte/character input with the most control for an application. It is similar to cbreak() mode, but also disables signal character processing (SIGINTR, SIGSUSP, SIGQUIT) and flow control processing (SIGSTART, SIGSTOP) so that the application can process them as it wants.

These modes affect all X/Open Curses input. The default input mode is inherited from the parent process when the application starts up.
A timeout similar to \texttt{halfdelay}(3XCURSES) can be applied to individual windows (see \texttt{timeout}(3XCURSES)). The \texttt{nodelay}(3XCURSES) function is equivalent to setting \texttt{wtimeout}(3XCURSES) for a window with a zero timeout (non-blocking) or infinite delay (blocking).

To handle function keys, \texttt{keypad}(3XCURSES) must be enabled. When it is enabled, the \texttt{getch()} function returns a \texttt{KEY\_constant} for a uniquely encoded key defined for that terminal. When \texttt{keypad()} is disabled, the \texttt{getch()} function returns the individual bytes composing the function key (see \texttt{getch}(3XCURSES) and \texttt{wget\_wch}(3XCURSES)). By default, \texttt{keypad()} is disabled.

When processing function keys, once the first byte is recognized, a timer is set for each subsequent byte in the sequence. If any byte in the function key sequence is not received before the timer expires, the bytes already received are pushed into a buffer and the original first byte is returned. Subsequent X/Open Curses input would take bytes from the buffer until exhausted, after which new input from the terminal will be requested. Enabling and disabling of the function key interbyte timer is handled by the \texttt{notimeout}(3XCURSES) function. By default, \texttt{notimeout()} is disabled (that is, the timer is used).

X/Open Curses always disables the terminal driver's echo processing. The \texttt{echo}(3XCURSES) and \texttt{noecho}(3XCURSES) functions control X/Open Curses software echoing. When software echoing is enabled, X/Open Curses input functions echo printable characters, control keys, and meta keys in the input window at the last cursor position. Functions keys are never echoed. When software echoing is disabled, it is the application's responsibility to handle echoing.

### EXAMPLES

**EXAMPLE 1**Copying Single-Column Characters Over Single-Column Characters

In the upcoming examples, some characters have special meanings:

- \{, [, and ( represent the left halves of multi-column characters. }, ], and ) represent the corresponding right halves of the same multi-column characters.
- Alphanumeric characters and periods (.) represent single-column characters.
- The number sign (#) represents the background character.

```c

Example 1: Copying Single-Column Characters Over Single-Column Characters

\begin{align*}
\text{copywin}(s, t, 0, 0, 1, 1, 0, 1, 3, 0) \Rightarrow t \\
\text{abcdef} & \rightarrow \text{bcd..} \\
\text{ghijkl} & \rightarrow \text{hij..}
\end{align*}

There are no special problems with this situation.
```

**EXAMPLE 2**Copying Multi-column Characters Over Single-Column Characters

```c

Example 2: Copying Multi-column Characters Over Single-Column Characters

\begin{align*}
\text{copywin}(s, t, 0, 0, 1, 1, 0, 1, 3, 0) \Rightarrow t \\
\text{a[]}\text{def} & \rightarrow \text{\[]}\text{d..}
\end{align*}
```
EXAMPLE 2 Copying Multi-column Characters Over Single-Column Characters

(Continued)

gh()kl ...... .h()..

There are no special problems with this situation.

EXAMPLE 3 Copying Single-Column Characters From Source Overlaps Multi-column Characters In Target

copywin(s, t, 0, 1, 0, 1, 1, 3, 0)

   s       t       →       t
   abcdef    []....    #bcd..
   ghijk tol    ...().    .hij#.

Overwriting multi-column characters in t has resulted in the # background characters being required to erase the remaining halves of the target's multi-column characters.

EXAMPLE 4 Copy Incomplete Multi-column Characters From Source To Target.

copywin(s, t, 0, 1, 0, 1, 1, 3, 0)

   s       t       →       t
   []cdef 123456    []cd56
   ghi()l 789012    7hi()2

The ] and ( halves of the multi-column characters have been copied from the source and expanded in the target outside of the specified target region.

Consider a pop-up dialog box that contains single-column characters and a base window that contains multi-column characters and you do the following:

save-dupwin(dialog); /* create backing store */
overwrite(cursor, save); /* save region to be overlayed */
wrefresh(dialog); /* display dialog */
wrefresh(save); /* restore screen image */
delwin(save); /* release backing store */

You can use code similar to this to implement generic popup() and popdown() routines in a variety of CURSES implementations (including BSD UNIX, and UNIX System V). In the simple case where the base window contains single-column characters only, it would correctly restore the image that appeared on the screen before the dialog box was displayed.

However, with multi-column characters, the overwrite() function might save a region with incomplete multi-column characters. The wrefresh(dialog) statement results in the behavior described in example 3 above. The behavior described in this example (that is, example 4) allows the wrefresh(save) statement to restore the window correctly.
EXAMPLE 5 Copying An Incomplete Multi-column Character To Region Next To Screen Margin (Not A Window Edge)

Two cases of copying an incomplete multi-column character to a region next to a screen margin follow:

\[ \text{copywin}(s, t, 0, 1, 0, 1, 2, 0) \]

\[
\begin{array}{c|c|c|c|c|c}
\text{s} & \text{t} & \rightarrow & \text{t} \\
\hline
[\] & cdef & 123456 & \#c456 \\
ghi & jkl & 789012 & hij012
\end{array}
\]

The background character (\#) replaces the \[ \] character that would have been copied from the source, because it is not possible to expand the multi-column character to its complete form.

\[ \text{copywin}(s, t, 0, 1, 0, 3, 1, 5, 0) \]

\[
\begin{array}{c|c|c|c|c|c}
\text{s} & \text{t} & \rightarrow & \text{t} \\
\hline
abcdef & 123456 & \text{123bcd} \\
ghi() & l & 789012 & 789хи\
\end{array}
\]

This second example is the same as the first, but with the right margin.

SEE ALSO

ksh(1), COLOR_PAIR(3XCURSES), PAIR_NUMBER(3XCURSES),
addchstr(3XCURSES), attr_off(3XCURSES), attroff(3XCURSES),
bkgdset(3XCURSES), bkgndset(3XCURSES), cbreak(3XCURSES),
copywin(3XCURSES), derwin(3XCURSES), echo(3XCURSES),
getcchar(3XCURSES), getch(3XCURSES), getnstr(3XCURSES),
halfdelay(3XCURSES), inch(3XCURSES), keypad(3XCURSES),
newpad(3XCURSES), newwin(3XCURSES), nocbreak(3XCURSES),
nodelay(3XCURSES), noecho(3XCURSES), noraw(3XCURSES),
notimeout(3XCURSES), overlay(3XCURSES), overwrite(3XCURSES),
setcchar(3XCURSES), subwin(3XCURSES), timeout(3XCURSES),
waddchstr(3XCURSES), waddstr(3XCURSES), wcwidth(3C),
wget_wch(3XCURSES), winsch(3XCURSES), wnoutrefresh(3XCURSES),
wprintw(3XCURSES), wrefresh(3XCURSES), wtimeout(3XCURSES),
termio(7I),
environ(5)
curs_getch(3CURSES)

NAME
curs_getch, getch, wgetch, mvgetch, mvwgetch, ungetch
characters from curses terminal keyboard

SYNOPSIS
cc [ fl... ] file ... -lcurses [ library ... ]
#include <curses.h>

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);

DESCRIPTION
With the getch(), wgetch(), mvgetch(), and mvwgetch() routines a character is
read from the terminal associated with the window. In no-delay mode, if no input is
waiting, the value ERR is returned. In delay mode, the program waits until the system
passes text through to the program. Depending on the setting of cbreak(), this is
after one character (cbreak mode), or after the first newline (nocbreak mode). In
half-delay mode, the program waits until a character is typed or the specified timeout
has been reached. Unless noecho() has been set, the character will also be echoed
into the designated window.

If the window is not a pad, and it has been moved or modified since the last call to
wrefresh(), wrefresh() will be called before another character is read.

If keypad() is TRUE, and a function key is pressed, the token for that function key is
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 a
character that could be the beginning of a function key (such as escape) is received,
curses sets a timer. If the remainder of the sequence does not come in within the
designated time, the character is passed through; otherwise, the function key value is
returned. For this reason, many terminals experience a delay between the time a user
presses the escape key and the escape is returned to the program. Since tokens
returned by these routines are outside the ASCII range, they are not printable.

The ungetch() routine places ch back onto the input queue to be returned by the
next call to wgetch().

Function Keys
The following function keys, defined in <curses.h>, might be returned by getch() if
keypad() has been enabled. Note that not all of these may be supported on a
particular terminal if the terminal does not transmit a unique code when the key is
pressed or if the definition for the key is not present in the terminfo database.

<table>
<thead>
<tr>
<th>Name</th>
<th>Key name</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_BREAK</td>
<td>Break key</td>
</tr>
<tr>
<td>Name</td>
<td>Key name</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>KEY_DOWN</td>
<td>The four arrow keys . . .</td>
</tr>
<tr>
<td>KEY_UP</td>
<td></td>
</tr>
<tr>
<td>KEY_LEFT</td>
<td></td>
</tr>
<tr>
<td>KEY_RIGHT</td>
<td></td>
</tr>
<tr>
<td>KEY_HOME</td>
<td>Home key (upward+left arrow)</td>
</tr>
<tr>
<td>KEY_BACKSPACE</td>
<td>Backspace</td>
</tr>
<tr>
<td>KEY_F0</td>
<td>Function keys; space for 64 keys is reserved.</td>
</tr>
<tr>
<td>KEY_F(n)</td>
<td>For $0 \leq n \leq 63$</td>
</tr>
<tr>
<td>KEY_DL</td>
<td>Delete line</td>
</tr>
<tr>
<td>KEY_IL</td>
<td>Insert line</td>
</tr>
<tr>
<td>KEY_DC</td>
<td>Delete character</td>
</tr>
<tr>
<td>KEY_IC</td>
<td>Insert char or enter insert mode</td>
</tr>
<tr>
<td>KEY_EIC</td>
<td>Exit insert char mode</td>
</tr>
<tr>
<td>KEY_CLEAR</td>
<td>Clear screen</td>
</tr>
<tr>
<td>KEY_EOS</td>
<td>Clear to end of screen</td>
</tr>
<tr>
<td>KEY_EOL</td>
<td>Clear to end of line</td>
</tr>
<tr>
<td>KEY_SF</td>
<td>Scroll 1 line forward</td>
</tr>
<tr>
<td>KEY_SR</td>
<td>Scroll 1 line backward (reverse)</td>
</tr>
<tr>
<td>KEY_NPAGE</td>
<td>Next page</td>
</tr>
<tr>
<td>KEY_PPAGE</td>
<td>Previous page</td>
</tr>
<tr>
<td>KEY_STAB</td>
<td>Set tab</td>
</tr>
<tr>
<td>KEY_CTAB</td>
<td>Clear tab</td>
</tr>
<tr>
<td>KEY_CATAB</td>
<td>Clear all tabs</td>
</tr>
<tr>
<td>KEY_ENTER</td>
<td>Enter or send</td>
</tr>
<tr>
<td>KEY_SRESET</td>
<td>Soft (partial) reset</td>
</tr>
<tr>
<td>KEY_RESET</td>
<td>Reset or hard reset</td>
</tr>
<tr>
<td>KEY_PRINT</td>
<td>Print or copy</td>
</tr>
<tr>
<td>KEY_LL</td>
<td>Home down or bottom (lower left). Keypad is</td>
</tr>
<tr>
<td></td>
<td>arranged like this: (Row 1) A1 up A3 (Row 2)</td>
</tr>
<tr>
<td></td>
<td>left B2 right (Row 3) C1 down C3</td>
</tr>
</tbody>
</table>
### curs_getch(3CURSES)

<table>
<thead>
<tr>
<th>Name</th>
<th>Key name</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_A1</td>
<td>Upper left of keypad</td>
</tr>
<tr>
<td>KEY_A3</td>
<td>Upper right of keypad</td>
</tr>
<tr>
<td>KEY_B2</td>
<td>Center of keypad</td>
</tr>
<tr>
<td>KEY_C1</td>
<td>Lower left of keypad</td>
</tr>
<tr>
<td>KEY_C3</td>
<td>Lower right of keypad</td>
</tr>
<tr>
<td>KEY_BTAB</td>
<td>Back tab key</td>
</tr>
<tr>
<td>KEY_BEG</td>
<td>Beg(inning) key</td>
</tr>
<tr>
<td>KEY_CANCEL</td>
<td>Cancel key</td>
</tr>
<tr>
<td>KEY_CLOSE</td>
<td>Close key</td>
</tr>
<tr>
<td>KEY_COMMAND</td>
<td>Cmd (command) key</td>
</tr>
<tr>
<td>KEY_COPY</td>
<td>Copy key</td>
</tr>
<tr>
<td>KEY_CREATE</td>
<td>Create key</td>
</tr>
<tr>
<td>KEY_END</td>
<td>End key</td>
</tr>
<tr>
<td>KEY_EXIT</td>
<td>Exit key</td>
</tr>
<tr>
<td>KEY_FIND</td>
<td>Find key</td>
</tr>
<tr>
<td>KEY_HELP</td>
<td>Help key</td>
</tr>
<tr>
<td>KEY_MARK</td>
<td>Mark key</td>
</tr>
<tr>
<td>KEY_MESSAGE</td>
<td>Message key</td>
</tr>
<tr>
<td>KEY_MOVE</td>
<td>Move key</td>
</tr>
<tr>
<td>KEY_NEXT</td>
<td>Next object key</td>
</tr>
<tr>
<td>KEY_OPEN</td>
<td>Open key</td>
</tr>
<tr>
<td>KEY_OPTIONS</td>
<td>Options key</td>
</tr>
<tr>
<td>KEY_PREVIOUS</td>
<td>Previous object key</td>
</tr>
<tr>
<td>KEY_REDO</td>
<td>Redo key</td>
</tr>
<tr>
<td>KEY_REFERENCE</td>
<td>Reference key</td>
</tr>
<tr>
<td>KEY_REFRESH</td>
<td>Refresh key</td>
</tr>
<tr>
<td>KEY_REPLACE</td>
<td>Replace key</td>
</tr>
<tr>
<td>KEY_RESTART</td>
<td>Restart key</td>
</tr>
<tr>
<td>KEY_RESUME</td>
<td>Resume key</td>
</tr>
</tbody>
</table>

Curses Library Functions  107
### curs_getch(3CURSES)

<table>
<thead>
<tr>
<th>Name</th>
<th>Key name</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_SAVE</td>
<td>Save key</td>
</tr>
<tr>
<td>KEY_SBEG</td>
<td>Shifted beginning key</td>
</tr>
<tr>
<td>KEY_SCANCEL</td>
<td>Shifted cancel key</td>
</tr>
<tr>
<td>KEY_SCOMMAND</td>
<td>Shifted command key</td>
</tr>
<tr>
<td>KEY_SCOPY</td>
<td>Shifted copy key</td>
</tr>
<tr>
<td>KEY_SCREATE</td>
<td>Shifted create key</td>
</tr>
<tr>
<td>KEY_SDC</td>
<td>Shifted delete char key</td>
</tr>
<tr>
<td>KEY(SDL)</td>
<td>Shifted delete line key</td>
</tr>
<tr>
<td>KEY_SELECT</td>
<td>Select key</td>
</tr>
<tr>
<td>KEY_SEND</td>
<td>Shifted end key</td>
</tr>
<tr>
<td>KEY_SEOL</td>
<td>Shifted clear line key</td>
</tr>
<tr>
<td>KEY_SEXIT</td>
<td>Shifted exit key</td>
</tr>
<tr>
<td>KEY_SFIN</td>
<td>Shifted find key</td>
</tr>
<tr>
<td>KEY_SHELP</td>
<td>Shifted help key</td>
</tr>
<tr>
<td>KEY_SHOME</td>
<td>Shifted home key</td>
</tr>
<tr>
<td>KEY_SIC</td>
<td>Shifted input key</td>
</tr>
<tr>
<td>KEY_SLEFT</td>
<td>Shifted left arrow key</td>
</tr>
<tr>
<td>KEY_SMESSAGE</td>
<td>Shifted message key</td>
</tr>
<tr>
<td>KEY_SMOVE</td>
<td>Shifted move key</td>
</tr>
<tr>
<td>KEY_SNEXT</td>
<td>Shifted next key</td>
</tr>
<tr>
<td>KEY_SOPTIONS</td>
<td>Shifted options key</td>
</tr>
<tr>
<td>KEY_SPREVIOUS</td>
<td>Shifted prev key</td>
</tr>
<tr>
<td>KEY_SPRINT</td>
<td>Shifted print key</td>
</tr>
<tr>
<td>KEY_SREDO</td>
<td>Shifted redo key</td>
</tr>
<tr>
<td>KEY_SREPLACE</td>
<td>Shifted replace key</td>
</tr>
<tr>
<td>KEY_SRIGHT</td>
<td>Shifted right arrow</td>
</tr>
<tr>
<td>KEY SRSUME</td>
<td>Shifted resume key</td>
</tr>
<tr>
<td>KEY_SSAVE</td>
<td>Shifted save key</td>
</tr>
<tr>
<td>KEY_SSUSPEND</td>
<td>Shifted suspend key</td>
</tr>
</tbody>
</table>
All routines return the integer `ERR` upon failure. The `ungetch()` routine returns an integer value other than `ERR` upon successful completion. The other routines return the next input character or function key code upon successful completion.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

### See Also

`curs_inopts(3CURSES), curs_move(3CURSES), curs_refresh(3CURSES), curses(3CURSES), attributes(5)`

### Notes

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Use of the escape key for a single character function is discouraged.

When using `getch()`, `wgetch()`, `mvgetch()`, or `mvwgetch()`, nocbreak mode (`nocbreak()`) and echo mode (`echo()`) should not be used at the same time. Depending on the state of the tty driver when each character is typed, the program may produce undesirable results.

Note that `getch()`, `mvgetch()`, and `mvwgetch()` may be macros.
NAME
curs_getstr, getstr, wgetstr, mvgetstr, mvwgetstr, wgetnstr – get character strings from
curses terminal keyboard

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

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 wgetnstr(WINDOW *win, char *str, 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 the character pointer str. wgetnstr() reads at most \textit{n} characters, thus
preventing a possible overflow of the input buffer. The user’s erase and kill characters
are interpreted, as well as any special keys (such as function keys, HOME key, and
CLEAR key.)

RETURN VALUES
All routines return the integer ERR upon failure and an integer value other than ERR
upon successful completion.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_getkey(3CURSES), curses(3CURSES), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and
<unctrl.h>.

Note that getstr(), mvgetstr(), and mvwgetstr() may be macros.
NAME
curs_getwch, getch, wgetch, mvgetch, mvwgetch, ungetch – get (or push back) wchar_t characters from curses terminal keyboard

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int getch(void);
int wgetch(WINDOW *win);
int mvgetch(int y, int x);
int mvwgetch(WINDOW *win, int y, int x);
int ungetch(int wch);

DESCRIPTION
The getch(), wgetch(), mvgetch(), and mvwgetch() routines read an EUC character from the terminal associated with the window, transform it into a wchar_t character, and return a wchar_t character. In no-delay mode, if no input is waiting, the value ERR is returned. In delay mode, the program waits until the system passes text through to the program. Depending on the setting of cbreak, this is after one character (cbreak mode), or after the first newline (nocbreak mode). In half-delay mode, the program waits until a character is typed or the specified timeout has been reached. Unless noecho has been set, the character will also be echoed into the designated window.

If the window is not a pad, and it has been moved or modified since the last call to wrefresh(3CURSES), wrefresh will be called before another character is read.

If keypad is TRUE, and a function key is pressed, the token for that function key is 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 a character that could be the beginning of a function key (such as escape) is received, curses(3CURSES) sets a timer. If the remainder of the sequence does not come in within the designated time, the character is passed through; otherwise, the function key value is returned. For this reason, many terminals experience a delay between the time a user presses the escape key and the escape is returned to the program.

The ungetch() routine places wch back onto the input queue to be returned by the next call to wgetch().

Function Keys
The following function keys, defined in <curses.h>, might be returned by getch() if keypad has been enabled. Note that not all of these may be supported on a particular terminal if the terminal does not transmit a unique code when the key is pressed or if the definition for the key is not present in the terminfo(4) database.

<table>
<thead>
<tr>
<th>Name</th>
<th>Key name</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_BREAK</td>
<td>Break key</td>
</tr>
<tr>
<td>Name</td>
<td>Key name</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>KEY_DOWN</td>
<td>The four arrow keys . . .</td>
</tr>
<tr>
<td>KEY_UP</td>
<td></td>
</tr>
<tr>
<td>KEY_LEFT</td>
<td></td>
</tr>
<tr>
<td>KEY_RIGHT</td>
<td></td>
</tr>
<tr>
<td>KEY_HOME</td>
<td>Home key (upward+left arrow)</td>
</tr>
<tr>
<td>KEY_BACKSPACE</td>
<td>Backspace</td>
</tr>
<tr>
<td>KEY_F0</td>
<td>Function keys; space for 64 keys is reserved.</td>
</tr>
<tr>
<td>KEY_F(n)</td>
<td>For $0 \leq n \leq 63$</td>
</tr>
<tr>
<td>KEY_DL</td>
<td>Delete line</td>
</tr>
<tr>
<td>KEY_IL</td>
<td>Insert line</td>
</tr>
<tr>
<td>KEY_DC</td>
<td>Delete character</td>
</tr>
<tr>
<td>KEY_IC</td>
<td>Insert char or enter insert mode</td>
</tr>
<tr>
<td>KEY_EIC</td>
<td>Exit insert char mode</td>
</tr>
<tr>
<td>KEY_CLEAR</td>
<td>Clear screen</td>
</tr>
<tr>
<td>KEY_EOS</td>
<td>Clear to end of screen</td>
</tr>
<tr>
<td>KEY_EOL</td>
<td>Clear to end of line</td>
</tr>
<tr>
<td>KEY_SF</td>
<td>Scroll 1 line forward</td>
</tr>
<tr>
<td>KEY_SR</td>
<td>Scroll 1 line backward (reverse)</td>
</tr>
<tr>
<td>KEY_NPAGE</td>
<td>Next page</td>
</tr>
<tr>
<td>KEY_PPAGE</td>
<td>Previous page</td>
</tr>
<tr>
<td>KEY_STAB</td>
<td>Set tab</td>
</tr>
<tr>
<td>KEY_CTAB</td>
<td>Clear tab</td>
</tr>
<tr>
<td>KEY_CATAB</td>
<td>Clear all tabs</td>
</tr>
<tr>
<td>KEY_ENTER</td>
<td>Enter or send</td>
</tr>
<tr>
<td>KEY_SRESET</td>
<td>Soft (partial) reset</td>
</tr>
<tr>
<td>KEY_RESET</td>
<td>Reset or hard reset</td>
</tr>
<tr>
<td>KEY_PRINT</td>
<td>Print or copy</td>
</tr>
<tr>
<td>KEY_LL</td>
<td>Home down or bottom (lower left). Keypad is arranged like this: A1 up A3 left B2 right C1 down C3</td>
</tr>
<tr>
<td>Name</td>
<td>Key name</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>KEY_A1</td>
<td>Upper left of keypad</td>
</tr>
<tr>
<td>KEY_A3</td>
<td>Upper right of keypad</td>
</tr>
<tr>
<td>KEY_B2</td>
<td>Center of keypad</td>
</tr>
<tr>
<td>KEY_C1</td>
<td>Lower left of keypad</td>
</tr>
<tr>
<td>KEY_C3</td>
<td>Lower right of keypad</td>
</tr>
<tr>
<td>KEY_BTAB</td>
<td>Back tab key</td>
</tr>
<tr>
<td>KEY_BEG</td>
<td>Beg(inning) key</td>
</tr>
<tr>
<td>KEY_CANCEL</td>
<td>Cancel key</td>
</tr>
<tr>
<td>KEY_CLOSE</td>
<td>Close key</td>
</tr>
<tr>
<td>KEY_COMMAND</td>
<td>Cmd (command) key</td>
</tr>
<tr>
<td>KEY_COPY</td>
<td>Copy key</td>
</tr>
<tr>
<td>KEY_CREATE</td>
<td>Create key</td>
</tr>
<tr>
<td>KEY_END</td>
<td>End key</td>
</tr>
<tr>
<td>KEY_EXIT</td>
<td>Exit key</td>
</tr>
<tr>
<td>KEY_FIND</td>
<td>Find key</td>
</tr>
<tr>
<td>KEY_HELP</td>
<td>Help key</td>
</tr>
<tr>
<td>KEY_MARK</td>
<td>Mark key</td>
</tr>
<tr>
<td>KEY_MESSAGE</td>
<td>Message key</td>
</tr>
<tr>
<td>KEY_MOVE</td>
<td>Move key</td>
</tr>
<tr>
<td>KEY_NEXT</td>
<td>Next object key</td>
</tr>
<tr>
<td>KEY_OPEN</td>
<td>Open key</td>
</tr>
<tr>
<td>KEY_OPTIONS</td>
<td>Options key</td>
</tr>
<tr>
<td>KEY_PREVIOUS</td>
<td>Previous object key</td>
</tr>
<tr>
<td>KEY_REDO</td>
<td>Redo key</td>
</tr>
<tr>
<td>KEY_REFERENCE</td>
<td>Reference key</td>
</tr>
<tr>
<td>KEY_REFRESH</td>
<td>Refresh key</td>
</tr>
<tr>
<td>KEY_REPLACE</td>
<td>Replace key</td>
</tr>
<tr>
<td>KEY_RESTART</td>
<td>Restart key</td>
</tr>
<tr>
<td>KEY_RESUME</td>
<td>Resume key</td>
</tr>
</tbody>
</table>

Curses Library Functions  113
<table>
<thead>
<tr>
<th>Name</th>
<th>Key name</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_SAVE</td>
<td>Save key</td>
</tr>
<tr>
<td>KEY_SBEG</td>
<td>Shfted beginning key</td>
</tr>
<tr>
<td>KEY_SCANCEL</td>
<td>Shfted cancel key</td>
</tr>
<tr>
<td>KEY_SCOMMAND</td>
<td>Shfted command key</td>
</tr>
<tr>
<td>KEY_SCOPY</td>
<td>Shfted copy key</td>
</tr>
<tr>
<td>KEY_SCRCREATE</td>
<td>Shfted create key</td>
</tr>
<tr>
<td>KEY_SDC</td>
<td>Shfted delete char key</td>
</tr>
<tr>
<td>KEY(SDL)</td>
<td>Shfted delete line key</td>
</tr>
<tr>
<td>KEY_SELECT</td>
<td>Select key</td>
</tr>
<tr>
<td>KEY_SEND</td>
<td>Shfted end key</td>
</tr>
<tr>
<td>KEY_SEOL</td>
<td>Shfted clear line key</td>
</tr>
<tr>
<td>KEY_SEXIT</td>
<td>Shfted exit key</td>
</tr>
<tr>
<td>KEY_SFIND</td>
<td>Shfted find key</td>
</tr>
<tr>
<td>KEY_SHELP</td>
<td>Shfted help key</td>
</tr>
<tr>
<td>KEY_SHOME</td>
<td>Shfted home key</td>
</tr>
<tr>
<td>KEY_SIC</td>
<td>Shfted input key</td>
</tr>
<tr>
<td>KEY_SLEFT</td>
<td>Shfted left arrow key</td>
</tr>
<tr>
<td>KEY_SMESSAGE</td>
<td>Shfted message key</td>
</tr>
<tr>
<td>KEY_SMOVE</td>
<td>Shfted move key</td>
</tr>
<tr>
<td>KEY_SNEXT</td>
<td>Shfted next key</td>
</tr>
<tr>
<td>KEY_SOPTIONS</td>
<td>Shfted options key</td>
</tr>
<tr>
<td>KEY_SPREVIOUS</td>
<td>Shfted prev key</td>
</tr>
<tr>
<td>KEY_SPRINT</td>
<td>Shfted print key</td>
</tr>
<tr>
<td>KEY_SREDO</td>
<td>Shfted redo key</td>
</tr>
<tr>
<td>KEY_SREPLACE</td>
<td>Shfted replace key</td>
</tr>
<tr>
<td>KEY_SRIGHT</td>
<td>Shfted right arrow</td>
</tr>
<tr>
<td>KEY_SRSUME</td>
<td>Shfted resume key</td>
</tr>
<tr>
<td>KEY_SSAVE</td>
<td>Shfted save key</td>
</tr>
<tr>
<td>KEY_SSUSPEND</td>
<td>Shfted suspend key</td>
</tr>
</tbody>
</table>
curs_getwch(3CURSES)

<table>
<thead>
<tr>
<th>Name</th>
<th>Key name</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_SUNDO</td>
<td>Shifted undo key</td>
</tr>
<tr>
<td>KEY_SUSPEND</td>
<td>Suspend key</td>
</tr>
<tr>
<td>KEY UNDO</td>
<td>Undo key</td>
</tr>
</tbody>
</table>

RETURN VALUE
All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), curs_inopts(3CURSES), curs_move(3CURSES),
wrefresh(3CURSES), terminfo(4), attributes(5)

NOTES
The header file <curses.h> automatically includes the header files <stdio.h>,
<unctrl.h> and <widec.h>.

Use of the escape key by a programmer for a single character function is discouraged.

When using getwch(), wgetwch(), mvgetwch(), or mvwgetwch(), nocbreak mode and echo mode should not be used at the same time. Depending on the state of the tty driver when each character is typed, the program may produce undesirable results.

Note that getwch(), mvgetwch(), and mvwgetwch() may be macros.
cur_getwstr(3CURSES)

NAME  
curs_getwstr, getwstr, getnwstr, wgetwstr, wgetnwstr, mvgetwstr, mvgetnwstr, 
mvwgetwstr, mvwgetnwstr – get wchar_t character strings from curses terminal 
keyboard

SYNOPSIS  
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int getwstr(wchar_t *wstr);
int getnwstr(wchar_t *wstr, int n);
int wgetwstr(WINDOW *win, wchar_t *wstr);
int wgetnwstr(WINDOW *win, wchar_t *wstr, int n);
int mvgetwstr(int y, int x, wchar_t *wstr);
int mvgetnwstr(int y, int x, wchar_t *wstr, int n);
int mvwgetwstr(WINDOW *win, int y, int x, wchar_t *wstr);
int mvwgetnwstr(WINDOW *win, int y, int x, wchar_t *wstr, int n);

DESCRIPTION  
The effect of getwstr() is as though a series of calls to getwch(3CURSES) were 
made, until a newline and carriage return is received. The resulting value is placed in 
the area pointed to by the wchar_t pointer wstr. getnwstr() reads at most n 
wchar_t characters, thus preventing a possible overflow of the input buffer. The 
user’s erase and kill characters are interpreted, as well as any special keys (such as 
function keys, HOME key, CLEAR key, etc.).

RETURN VALUE  
All routines return the integer ERR upon failure and an integer value other than ERR 
upon successful completion.

ATTRIBUTES  
See attributes(5) for a description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curses(3CURSES), getwch(3CURSES), attributes(5)

NOTES  
The header file <curses.h> automatically includes the header files <stdio.h>, 
<unctrl.h>, and <widec.h>.

Note that all routines except wgetnwstr() may be macros.
curses_getyx(3CURSES)

NAME  
curs_getyx, getyx, getparyx, getbegyx, getmaxyx – get curses cursor and window coordinates

SYNOPSIS  
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

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);

DESCRIPTION  
With the getyx() macro, the cursor position of the window is placed in the two integer variables y and x.

With the getparyx() macro, if win is a subwindow, the beginning coordinates of the subwindow relative to the parent window are placed into two integer variables, y and x. Otherwise, -1 is placed into y and x.

Like getyx(), the getbegyx() and getmaxyx() macros store the current beginning coordinates and size of the specified window.

RETURN VALUES  
The return values of these macros are undefined (that is, they should not be used as the right-hand side of assignment statements).

ATTRIBUTES  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curses(3CURSES), attributes(5)

NOTES  
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.

Note that all of these interfaces are macros and that "&" is not necessary before the variables y and x.
NAME
curs_inch, inch, winch, mvinch, mvwinch – get a character and its attributes from a
curses window

SYNOPSIS
c { flag ... } file ... -lcurses [ library ... ]
#include <curses.h>

chtype inch(void);
chtype winch(WINDOW *win);
chtype mvinch(int y, int x);
chtype mvwinch(WINDOW *win, int y, int x);

DESCRIPTION
With these routines, the character, of type chtype, at the current position in the
named window is returned. If any attributes are set for that position, their values are
OR-ed into the value returned. Constants defined in <curses.h> can be used with
the logical AND (&) operator to extract the character or attributes alone.

Attributes
The following bit-masks may be AND-ed with characters returned by winch().

A_CHARTEXT Bit-mask to extract character
A_ATTRIBUTES Bit-mask to extract attributes
A_COLOR Bit-mask to extract color-pair field information

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and
<unctrl.h>.

Note that all of these routines may be macros.
NAME    curs_inchstr, inchstr, inchnstr, winchstr, winchnstr, mvwinchstr, mvwinchnstr, mvwinchnstr – get a string of characters (and attributes) from a curses window

SYNOPSIS    cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int inchstr(chtype *chstr);
int inchnstr(chtype *chstr, int n);
int winchstr(WINDOW *win, chtype *chstr);
int winchnstr(WINDOW *win, chtype *chstr, int n);
int mvinchstr(int y, int x, chtype *chstr);
int mvwinchnstr(int y, int x, chtype *chstr, int n);
int mwwinchstr(WINDOW *win, int y, int x, chtype *chstr);
int mwwinchnstr(WINDOW *win, int y, int x, chtype *chstr, int n);

DESCRIPTION    With these routines, a string of type chtype, starting at the current cursor position in
the named window and ending at the right margin of the window, is returned. The
four functions with n as the last argument, return the string at most n characters long.
Constants defined in <curses.h> can be used with the & (logical AND) operator to
extract the character or the attribute alone from any position in the chstr (see
curs_inch(3CURSES)).

RETURN VALUES    All routines return the integer ERR upon failure and an integer value other than ERR
upon successful completion.

ATTRIBUTES    See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO    curs_inch(3CURSES), curses(3CURSES), attributes(5)

NOTES    The header <curses.h> automatically includes the headers <stdio.h> and
<unctrl.h>.

Note that all routines except winchnstr() may be macros.
## NAME

curs_initscr, initscr, newterm, endwin, isendwin, set_term, delscreen – curses screen initialization and manipulation routines

## SYNOPSIS

```c
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
WINDOW *initscr(void);
int endwin(void);
int isendwin(void);
SCREEN *newterm(char *type, FILE *outfd, FILE *infd);
SCREEN *set_term(SCREEN *new);
void delscreen(SCREEN *sp);
```

## DESCRIPTION

`initscr()` is almost always the first routine that should be called (the exceptions are `slk_init()`, `filter()`, `ripoffline()`, `use_env()` and, for multiple-terminal applications, `newterm()`). This determines the terminal type and initializes all curses data structures. `initscr()` also causes the first call to `refresh()` to clear the screen. If errors occur, `initscr()` writes an appropriate error message to standard error and exits; otherwise, a pointer is returned to `stdscr()`. If the program needs an indication of error conditions, `newterm()` should be used instead of `initscr()`. `initscr()` should only be called once per application.

A program that outputs to more than one terminal should use the `newterm()` routine for each terminal instead of `initscr()`. A program that needs an indication of error conditions, so it can continue to run in a line-oriented mode if the terminal cannot support a screen-oriented program, would also use this routine. The routine `newterm()` should be called once for each terminal. It returns a variable of type `SCREEN *` which should be saved as a reference to that terminal. The arguments are the `type` of the terminal to be used in place of `$TERM`, a file pointer for output to the terminal, and another file pointer for input from the terminal (if `type` is `NULL`, `$TERM` will be used). The program must also call `endwin()` for each terminal being used before exiting from curses. If `newterm()` is called more than once for the same terminal, the first terminal referred to must be the last one for which `endwin()` is called.

A program should always call `endwin()` before exiting or escaping from curses mode temporarily. This routine restores tty modes, moves the cursor to the lower left-hand corner of the screen and resets the terminal into the proper non-visual mode. Calling `refresh()` or `doupdate()` after a temporary escape causes the program to resume visual mode.

The `isendwin()` routine returns `TRUE` if `endwin()` has been called without any subsequent calls to `wrefresh()`, and `FALSE` otherwise.

The `set_term()` routine is used to switch between different terminals. The screen reference `new` becomes the new current terminal. The previous terminal is returned by
The routine. This is the only routine which manipulates SCREEN pointers; all other routines affect only the current terminal.

The delscreen() routine frees storage associated with the SCREEN data structure. The endwin() routine does not do this, so delscreen() should be called after endwin() if a particular SCREEN is no longer needed.

**RETURN VALUES**

endwin() returns the integer ERR upon failure and OK upon successful completion.

Routines that return pointers always return NULL on error.

**ATTRIBUTES**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

curs_kernel(3CURSES), curs_refresh(3CURSES), curs_slk(3CURSES), curs_util(3CURSES), curses(3CURSES), attributes(5)

**NOTES**

The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.

Note that initscr() and newterm() may be macros.
NAME
curs_inopts, cbreak, nocbreak, echo, noecho, halfdelay, intrflush, keypad, meta, nodelay, notimeout, raw, noraw, noqiflush, qiflush, timeout, wtimeout, typeahead –
curses terminal input option control routines

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
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 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);

DESCRIPTION
The cbreak() and nocbreak() routines put the terminal into and out of cbreak() mode, respectively. In this mode, characters typed by the user are immediately available to the program, and erase/kill character-processing is not performed. When out of this mode, the tty driver buffers the typed characters 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 be in cbreak() mode, as the mode is inherited; therefore, a program should call cbreak() or nocbreak() explicitly. Most interactive programs using curses set the cbreak() mode.

Note that cbreak() overrides raw(). (See curs_getch(3CURSES) for a discussion of how these routines interact with echo() and noecho().)

The echo() and noecho() routines control whether characters typed by the user are echoed by getch() as they are typed. Echoing by the tty driver is always disabled, but initially getch() is in echo mode, so characters typed are echoed. Authors of
most interactive programs prefer to do their own echoing in a controlled area of the screen, or not to echo at all, so they disable echoing by calling `noecho()`. (See `curs_getch(3CURSES)` for a discussion of how these routines interact with `cbreak()` and `nocbreak()`.)

The `halfdelay()` routine is used for half-delay mode, which is similar to `cbreak()` mode in that characters typed by the user are immediately available to the program. However, after blocking for `tenths` tenths of seconds, `ERR` is returned if nothing has been typed. The value of `tenths` must be a number between 1 and 255. Use `nocb()` to leave half-delay mode.

If the `intrflush()` option is enabled, (`bf` is `TRUE`), when an interrupt key is pressed on the keyboard (interrupt, break, quit) all output in the tty driver queue will be flushed, giving the effect of faster response to the interrupt, but causing `curses` to have the wrong idea of what is on the screen. Disabling (`bf` is `FALSE`), the option prevents the flush. The default for the option is inherited from the tty driver settings. The window argument is ignored.

The `keypad()` option enables the keypad of the user’s terminal. If enabled (`bf` is `TRUE`), the user can press a function key (such as an arrow key) and `wgetch()` returns a single value representing the function key, as in `KEY_LEFT`. If disabled (`bf` is `FALSE`), `curses` does not treat function keys specially and the program has to interpret the escape sequences itself. If the keypad in the terminal can be turned on (made to transmit) and off (made to work locally), turning on this option causes the terminal keypad to be turned on when `wgetch()` is called. The default value for `keypad` is false.

Initially, whether the terminal returns 7 or 8 significant bits on input depends on the control mode of the tty driver (see `termio(7)`). To force 8 bits to be returned, invoke `meta(win, TRUE)`. To force 7 bits to be returned, invoke `meta(win, FALSE)`. The window argument, `win`, is always ignored. If the terminfo capabilities `smm` (`meta_on`) and `rmm` (`meta_off`) are defined for the terminal, `smm` is sent to the terminal when `meta(win, TRUE)` is called and `rmm` is sent when `meta(win, FALSE)` is called.

The `nodelay()` option causes `getch()` to be a non-blocking call. If no input is ready, `getch()` returns `ERR`. If disabled (`bf` is `FALSE`), `getch()` waits until a key is pressed.

While interpreting an input escape sequence, `wgetch()` sets a timer while waiting for the next character. If `notimeout(win, TRUE)` is called, then `wgetch()` does not set a timer. The purpose of the timeout is to differentiate between sequences received from a function key and those typed by a user.

With the `raw()` and `noraw()` routines, the terminal is 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 interrupt, quit, suspend, and flow control characters are all passed through uninterpreted, instead of generating a signal. The behavior of the BREAK key depends on other bits in the tty driver that are not set by `curses.`
When the noqiflush() routine is used, normal flush of input and output queues associated with the INTR, QUIT and SUSP characters will not be done (see termio(7I)). When qiflush() is called, the queues will be flushed when these control characters are read.

The timeout() and wtimeout() routines set blocking or non-blocking read for a given window. If delay is negative, blocking read is used (that is, waits indefinitely for input). If delay is zero, then non-blocking read is used (that is, read returns ERR if no input is waiting). If delay is positive, then read blocks for delay milliseconds, and returns ERR if there is still no input. Hence, these routines provide the same functionality as nodelay(), plus the additional capability of being able to block for only delay milliseconds (where delay is positive).

curses does "line-breakout optimization" by looking for typeahead periodically while updating the screen. If input is found, and it is coming from a tty, the current update is postponed until refresh() or doupdate() is called again. This allows faster response to commands typed in advance. Normally, the input FILE pointer passed to newterm(), or stdin in the case that initscr() was used, will be used to do this typeahead checking. The typeahead() routine specifies that the file descriptor fdedes is to be used to check for typeahead instead. If fdedes is -1, then no typeahead checking is done.

RETURN VALUES
All routines that return an integer return ERR upon failure and an integer value other than ERR upon successful completion, unless otherwise noted in the preceding routine descriptions.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
\[curs_getch(3CURSES), curs_initscr(3CURSES), curses(3CURSES), attributes(5), termio(7I)\]

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.

Note that echo(), noecho(), halfdelay(), intrflush(), meta(), nodelay(), notimeout(), noqiflush(), qiflush(), timeout(), and wtimeout() may be macros.
curs_insch, insch, winsch, mvinsch, mvwinsch – insert a character before the character under the cursor in a curses window

SYNOPSIS

```
cc [-f... -I... -lcurses ...] -I<...>
#include <curses.h>

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);
```

DESCRIPTION

With these routines, the character `ch` is inserted before the character under the cursor. All characters to the right of the cursor are moved one space to the right, with the possibility of the rightmost character on the line being lost. The cursor position does not change (after moving to `y`, `x`, if specified). (This does not imply use of the hardware insert character feature.)

RETURN VALUES

All routines return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

curses(3Curses), attributes(5)

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>.

Note that `insch()`, `mvinsch()`, and `mvwinsch()` may be macros.
NAME
curs_insstr, instr, insnstr, winsstr, winsnstr, mvinsstr, mvinsnstr, mvwinsstr,
mvwinsnstr – insert string before character under the cursor in a curses window

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int instr(char *str);
int insnstr(char *str, int n);
int winsstr(WINDOW *win, char *str);
int winsnstr(WINDOW *win, char *str, int n);
int mvinsstr(int y, int x, char *str);
int mvinsnstr(int y, int x, char *str, int n);
int mvwinsstr(WINDOW *win, int y, int x, char *str);
int mvwinsnstr(WINDOW *win, int y, int x, char *str, 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).
(This does not imply use of the hardware insert character feature.) The four routines
with n as the last argument insert at most n characters. If n<=0, then the entire string is
inserted.

If a character in str is a tab, newline, carriage return or backspace, the cursor is moved
appropriately within the window. A newline also does a clrtoeol() before moving.
Tabs are considered to be at every eighth column. If a character in str is another control
character, it is drawn in the ^X notation. Calling winch() after adding a control
character (and moving to it, if necessary) does not return the control character, but
instead returns the representation of the control character.

RETURN VALUES
All routines return the integer ERR upon failure and an integer value other than ERR
upon successful completion.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_clear(3CURSES), curs_inch(3CURSES), curses(3CURSES),
attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and
<unctrl.h>.
Note that all but `winsnstr()` may be macros.
curs_instr(3CURSES)

NAME
curs_instr, instr, innstr, winstr, winnstr, mvinstr, mvinnstr, mvwinstr, mvwinnstr – get a string of characters from a curses window

SYNOPSIS
c curses [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

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);

DESCRIPTION
These routines return a string of characters in str, starting at the current cursor position in the named window and ending at the right margin of the window. Attributes are stripped from the characters. The four functions with n as the last argument return the string at most n characters long.

RETURN VALUES
All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.

Note that all routines except winnstr() may be macros.
NAME | curs_inswch, inswch, winswch, mvinswch, mvwinswch – insert a wchar_t character before the character under the cursor in a curses window
SYNOPSIS | `cc [ flag ... ] file ... -lcurses [ library .. ]`
          #include <curses.h>
          int inswch(chtype wch);
          int winswch(WINDOW *win, chtype wch);
          int mvinswch(int y, int x, chtype wch);
          int mvwinswch(WINDOW *win, int y, int x, chtype wch);
DESCRIPTION | These routines insert the character \texttt{wch}, holding a wchar_t character, before the character under the cursor. All characters to the right of the cursor are moved one space to the right, with the possibility of the rightmost character on the line being lost. The cursor position does not change (after moving to \texttt{y, x}, if specified). (This does not imply use of the hardware insert character feature.)
RETURN VALUE | All routines return the integer \texttt{ERR} upon failure and an integer value other than \texttt{ERR} upon successful completion.
ATTRIBUTES | See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>
SEE ALSO | curses(3CURSES), attributes(5)
NOTES | The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h> and <widec.h>.

Note that \texttt{inswch()}, \texttt{mvinswch()}, and \texttt{mvwinswch()} may be macros.

None of these routines can use the color attribute in \texttt{chttype}. 

Curses Library Functions   129
NAME    curs_inswstr, inswstr, insnwstr, winswstr, winsnwstr, mvinswstr, mvinsnwstr, mvwinswstr, mvwinsnwstr – insert wchar_t string before character under the cursor in a curses window

SYNOPSIS  
```c
#include <curses.h>

int inswstr(wchar_t *wstr);
int insnwstr(wchar_t *wstr, int n);
int winswstr(WINDOW *win, wchar_t *wstr);
int winsnwstr(WINDOW *win, wchar_t *wstr, int n);
int mvinswstr(int y, int x, wchar_t *wstr);
int mvinsnwstr(int y, int x, wchar_t *wstr, int n);
int mvwinswstr(WINDOW *win, int y, int x, wchar_t *wstr);
int mvwinsnwstr(WINDOW *win, int y, int x, wchar_t *wstr, int n);
```

DESCRIPTION  These routines insert a wchar_t character string (as many wchar_t characters as will fit on the line) 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). (This does not imply use of the hardware insert character feature.) The four routines with n as the last argument insert at most n wchar_t characters. If n<=0, then the entire string is inserted.

If a character in wstr is a tab, newline, carriage return, or backspace, the cursor is moved appropriately within the window. A newline also does a clrtoeol(3CURSES) before moving. Tabs are considered to be at every eighth column. If a character in wstr is another control character, it is drawn in the ^X notation. Calling winwch(3CURSES) after adding a control character (and moving to it, if necessary) does not return the control character, but instead returns the representation of the control character.

RETURN VALUE  All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  clrtoeol(3CURSES), curses(3CURSES), winwch(3CURSES), attributes(5)

NOTES  The header file <curses.h> automatically includes the header files <stdio.h>, <unctrl.h> and <widec.h>.
Note that all but winsnwstr() may be macros.
curs_inwch(3CURSES)

NAME  
curs_inwch, inwch, winwch, mvinwch, mvwinwch -- get a wchar_t character and its
attributes from a curses window

SYNOPSIS  
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

chtype inwch(void);
chtype winwch(WINDOW *win);
chtype mvinwch(int y, int x);
chtype mvwinwch(WINDOW *win, int y, int x);

DESCRIPTION  
These routines return the wchar_t character, of type ctype, at the current position
in the named window. If any attributes are set for that position, their values are OR-ed
into the value returned. Constants defined in <curses.h> can be used with the
logical AND (&) operator to extract the character or attributes alone.

Attributes  
The following bit-masks may be AND-ed with characters returned by winwch().

A_WCHARTEXT Bit-mask to extract character
A_WATTRIBUTES Bit-mask to extract attributes

ATTRIBUTES  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curses(3CURSES), attributes(5)

NOTES  
The header file <curses.h> automatically includes the header files <stdio.h>,
<unctrl.h> and <widec.h>.

Note that all of these routines may be macros.

None of these routines can use the color attribute in ctype.
curs_inwchstr, inwchstr, inwchnstr, winwchstr, winwchnstr, mvinwchstr, mvinwchnstr, mvwinwchstr, mvwinwchnstr – get a string of \texttt{wchar_t} characters (and attributes) from a curses window

### SYNOPSIS

```c
#include <curses.h>

int inwchstr(chtype *wchstr);
int inwchnstr(chtype *wchstr, int n);
int winwchstr(WINDOW *win, chtype *wchstr);
int winwchnstr(WINDOW *win, chtype *wchstr, int n);
int mvinwchstr(int y, int x, chtype *wchstr);
int mvinwchnstr(int y, int x, chtype *wchstr, int n);
int mvwinwchstr(WINDOW *win, int y, int x, chtype *wchstr);
int mvwinwchnstr(WINDOW *win, int y, int x, chtype *wchstr, int n);
```

### DESCRIPTION

These routines return a string of type \texttt{chtpe}, holding \texttt{wchar_t} characters, starting at the current cursor position in the named window and ending at the right margin of the window. The four functions with \texttt{n} as the last argument, return the string at most \texttt{n} \texttt{wchar_t} characters long. Constants defined in \texttt{<curses.h>} can be used with the logical AND (\&) operator to extract the \texttt{wchar_t} character or the attribute alone from any position in the \texttt{wchstr} (see \texttt{curs_inwch(3CURSES)}).

### RETURN VALUE

All routines return the integer \texttt{ERR} upon failure and an integer value other than \texttt{ERR} upon successful completion.

### ATTRIBUTES

See \texttt{attributes(5)} for a description of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

### SEE ALSO

\texttt{curses(3CURSES), curs_inwch(3CURSES), attributes(5)}

### NOTES

The header file \texttt{<curses.h>} automatically includes the header files \texttt{<stdio.h>}, \texttt{<unctrl.h>} and \texttt{<widec.h>}.

Note that all routines except \texttt{winwchnstr()} may be macros.

None of these routines can use the color attribute in \texttt{chtpe}.
NAME
curs_inwstr, inwstr, innwstr, winwstr, winnwstr, mvinwstr, mvinnwstr, mvwinwstr, mvwinnwstr – get a string of wchar_t characters from a curses window

SYNOPSIS
c { flag ... } file ... -lcurses[library ... ]
#include <curses.h>

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 routines return the string of wchar_t characters in wstr starting at the current
cursor position in the named window and ending at the right margin of the window.
Attributes are stripped from the characters. The four functions with n as the last
argument return the string at most n wchar_t characters long.

RETURN VALUES
All routines return the integer ERR upon failure and an integer value other than ERR
upon successful completion.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), attributes(5)

NOTES
The header file <curses.h> automatically includes the header files <stdio.h>,
<unctrl.h> and <widec.h>.

Note that all routines except winnwstr() may be macros.
NAME

curs_kernel, def_prog_mode, def_shell_mode, reset_prog_mode, reset_shell_mode,
resetty, savetty, getsyx, setsyx, ripoffline, curs_set, napms – low-level curses routines

SYNOPSIS

cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

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);
int getsyx(int y, int x);
int setsyx(int y, int x);
int ripoffline(int line, int (*init)(WINDOW *, int));
int curs_set(int visibility);
int napms(int ms);

DESCRIPTION

The following routines give low-level access to various curses functionality. Theses
routines typically are used inside library routines.

The def_prog_mode() and def_shell_mode() routines 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() routines. This is done
automatically by initscr().

The reset_prog_mode() and reset_shell_mode() routines restore the terminal
to “program” (in curses) or “shell” (out of curses) state. These are done
automatically by endwin() and, after an endwin(), by doupdate(), so they
normally are not called.

The resetty() and savetty() routines 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().

With the getsyx() routine, the current coordinates of the virtual screen cursor are
returned in y and x. If leaveok() is currently TRUE, then −1,−1 is returned. If lines
have been removed from the top of the screen, using ripoffline(), y and x include
these lines; therefore, y and x should be used only as arguments for setsyx().

With the setsyx() routine, the virtual screen cursor is set to y, x. If y and x are both
−1, then leaveok() is set. The two routines getsyx() and setsyx() are designed
to be used by a library routine, which manipulates curses windows but does not
want to change the current position of the program’s cursor. The library routine would
call getyx() at the beginning, do its manipulation of its own windows, do a
wnoutrefresh() on its windows, call setsyx(), and then call doupdate().

The ripoffline() routine provides access to the same facility that slk_init()
(see curs_slk(3CURSES)) uses to reduce the size of the screen. ripoffline() must
be called before initscr() or newterm() is called. If line is positive, a line is
removed from the top of stdscr(); if line is negative, a line is removed from the
bottom. When this is done inside initscr(), the routine init() (supplied by the
user) is called with two arguments: a window pointer to the one-line window that has
been allocated and an integer with the number of columns in the window. Inside this
initialization routine, the integer variables LINES and COLS (defined in <curses.h>)
are not guaranteed to be accurate and wrefresh() or doupdate() must not be
called. It is allowable to call wnoutrefresh() during the initialization routine.

ripoffline() can be called up to five times before calling initscr() or
newterm().

With the curs_set() routine, the cursor state is set to invisible, normal, or very
visible for visibility equal to 0, 1, or 2 respectively. If the terminal supports the visibility
requested, the previous cursor state is returned; otherwise, ERR is returned.

The napms() routine is used to sleep for ms milliseconds.

RETURN VALUES
Except for curs_set(), these routines always return OK. curs_set() returns the
previous cursor state, or ERR if the requested visibility is not supported.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_initscr(3CURSES), curs_outopts(3CURSES), curs_refresh(3CURSES),
curs_scr_dump(3CURSES), curs_slk(3CURSES), curses(3CURSES),
attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and
<unctrl.h>.

Note that getyx() is a macro, so an ampersand (&) is not necessary before the
variables y and x.
NAME
curs_move, move, wmove – move curses window cursor

SYNOPSIS

cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

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

DESCRIPTION

With these routines, the cursor associated with the window is moved to line y and
column x. This routine does not move the physical cursor of the terminal until
refresh() is called. The position specified is relative to the upper left-hand corner of
the window, which is (0,0).

RETURN VALUES

These routines return the integer ERR upon failure and an integer value other than
ERR upon successful completion.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_refresh(3CURSES), curses(3CURSES), attributes(5)

NOTES

The header <curses.h> automatically includes the headers <stdio.h> and
<unctrl.h>.

Note that move() may be a macro.
NAME
curs_outopts, clearok, idlok, idcok, immedok, leaveok, setscrreg, wsetscrreg, scrollok, nl, nonl – curses terminal output option control routines

SYNOPSIS
ce [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
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 nl(void);
int nonl(void);

DESCRIPTION
These routines set options that deal with output within curses. All options are initially FALSE, unless otherwise stated. It is not necessary to turn these options off before calling endwin().

With the clearok() routine, if enabled (bf is TRUE), the next call to wrefresh() with this window will clear the screen completely and redraw the entire screen from scratch. This is useful when the contents of the screen are uncertain, or in some cases for a more pleasing visual effect. If the win argument to clearok() is the global variable curscr(), the next call to wrefresh() with any window causes the screen to be cleared and repainted from scratch.

With the idlok() routine, if enabled (bf is TRUE), curses considers using the hardware insert/delete line feature of terminals so equipped. If disabled (bf is FALSE), curses very seldom uses this feature. (The insert/delete character feature is always considered.) This option should be enabled only if the application needs insert/delete line, for example, for a screen editor. It is disabled by default because insert/delete line tends to be visually annoying when used in applications where it isn’t really needed. If insert/delete line cannot be used, curses redraws the changed portions of all lines.

With the idcok() routine, if enabled (bf is TRUE), curses considers using the hardware insert/delete character feature of terminals so equipped. This is enabled by default.

With the immedok() routine, if enabled (bf is TRUE), any change in the window image, such as the ones caused by waddch(), wclrtob(), wscr(), etc., automatically cause a call to wrefresh(). However, it may degrade the performance
curs_outopts(3CURSES)

considerably, due to repeated calls to wrefresh(). It is disabled by default. Normally, the hardware cursor is left at the location of the window cursor being refreshed. The leaveok() option 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() routines allow the application programmer to set a software scrolling region in a window. 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 line causes all lines in the scrolling region to scroll up one line. Only the text of the window is scrolled. (Note that this has nothing to do with the use of a physical scrolling region capability in the terminal, like that in the VT100. If idlok() is enabled and the terminal has either a scrolling region or insert/delete line capability, they will probably be used by the output routines.)

The scrollok() option controls what happens when the cursor of a window is moved off the edge of the window or scrolling region, either as a result of a newline action on the bottom line, or typing the last character of the last line. If disabled, (bf is FALSE), the cursor is left on the bottom line. If enabled, (bf is TRUE), wrefresh() is called on the window, and the physical terminal and window are scrolled up one line. (Note that in order to get the physical scrolling effect on the terminal, it is also necessary to call idlok().)

The nl() and nonl() routines control whether newline is translated into carriage return and linefeed on output, and whether return is translated into newline on input. Initially, the translations do occur. By disabling these translations using nonl(), curses is able to make better use of the linefeed capability, resulting in faster cursor motion.

RETURN VALUES setscrreg() and wsetscrreg() return OK upon success and ERR upon failure. All other routines that return an integer always return OK.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curs_addch(3CURSES), curs_clear(3CURSES), curs_initscr(3CURSES), curs_refresh(3CURSES), curs_scroll(3CURSES), curses(3CURSES), attributes(5)

NOTES The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
Note that clearok(), leaveok(), scrollok(), idcok(), nl(), nonl(), and setscreqreg() may be macros.

The immedok() routine is useful for windows that are used as terminal emulators.
curs_overlay(3CURSES)

NAME
curs_overlay, overlay, overwrite, copywin – overlap and manipulate overlapped curses windows

SYNOPSIS
c { flag ... } file ... -lcurses { library ... }
#include <curses.h>

int overlay(WINDOW *srcwin, WINDOW *dstwin);
int overwrite(WINDOW *srcwin, WINDOW *dstwin);
int copywin(WINDOW *srcwin, WINDOW *dstwin, int sminrow, int smincol,
            int dminrow, int dmincol, int dmaxrow, int dmaxcol, int overlay);

DESCRIPTION
The overlay() and overwrite() routines overlay srcwin on top of dstwin. srcwin
and dstwin are not required to be the same size; only text where the two windows
overlap is copied. The difference is that overlay() is non-destructive (blanks are not
copied) whereas overwrite() is destructive.

The copywin() routine provides a finer granularity of control over the overlay()
and overwrite() routines. Like in the refresh() routine, a rectangle is specified
in the destination window, (dminrow, dmincol) and (dmaxrow, dmaxcol), and the
upper-left-corner coordinates of the source window, (sminrow, smincol). If the
argument overlay is true, then copying is non-destructive, as in overlay().

RETURN VALUES
Routines that return an integer return ERR upon failure and an integer value other
than ERR upon successful completion.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_pad(3CURSES), curs_refresh(3CURSES), curses(3CURSES),
attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and
<unctrl.h>.

Note that overlay() and overwrite may be macros.
curs_pad(3CURSES)

NAME  curs_pad, newpad, subpad, prefresh, pnoutrefresh, pechochar, pechowchar – create
       and display curses pads

SYNOPSIS  cc [ flag ... ] file ... -lcurses [ library ...]
#include <curses.h>

WINDOW *newpad(int nlines, int ncols);
WINDOW *subpad(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);

int prefresh(WINDOW *pad, int pminrow, int pmincol, int sminrow, int smincol, int smaxrow, int smaxcol);
int pnoutrefresh(WINDOW *pad, int pminrow, int pmincol, int sminrow, int smincol, int smaxrow, int smaxcol);

int pechochar(WINDOW *pad, chtype ch);
int pechowchar(WINDOW *pad, chtype wch);

DESCRIPTION  The newpad() routine creates and returns a pointer to a new pad data structure with
       the given number of lines, nlines, and columns, ncols. A pad is like a window, except
       that it is not restricted by the screen size, and is not necessarily associated with a
       particular part of the screen. Pads can be used when a large window is needed, and
       only a part of the window will be on the screen at one time. Automatic refreshes of
       pads (for example, from scrolling or echoing of input) do not occur. It is not legal to
call wrefresh(3CURSES) 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 the display.

The subpad() routine creates and returns a pointer to a subwindow within a pad
with the given number of lines, nlines, and columns, ncols. Unlike subwin(3CURSES),
which uses screen coordinates, the window is at position (begin_x, begin_y) on the
pad. The window is made in the middle of the window orig, so that changes made to
one window affect both windows. During the use of this routine, it will often be
necessary to call touchwin(3CURSES) or touchline(3CURSES) on orig before
calling prefresh().

The prefresh() and pnoutrefresh() routines are analogous to
wrefresh(3CURSES) and wnoutrefresh(3CURSES) except that they relate to pads
instead of windows. The additional parameters are needed to indicate what part of the
pad and screen are involved. pminrow and pmincol specify the upper left-hand corner
of the rectangle to be displayed in the pad. sminrow, smincol, smaxrow, and smaxcol
specify the edges of the rectangle to be displayed on the screen. The lower right-hand
corner of the rectangle to be displayed in the pad is calculated from the screen
coordinates, since the rectangles must be the same size. Both rectangles must be
entirely contained within their respective structures. Negative values of pminrow,
pmincol, sminrow, or smincol are treated as if they were zero.
The `pechochar()` routine is functionally equivalent to a call to `addch(3CURSES)` followed by a call to `refresh(3CURSES)`, a call to `waddch(3CURSES)` followed by a call to `wrefresh(3CURSES)`, or a call to `waddch(3CURSES)` followed by a call to `prefresh()`. The knowledge that only a single character is being output is taken into consideration and, for non-control characters, a considerable performance gain might be seen by using these routines instead of their equivalents. In the case of `pechochar()`, the last location of the pad on the screen is reused for the arguments to `prefresh()`.

**RETURN VALUES**

Routines that return an integer return `ERR` upon failure and an integer value other than `ERR` upon successful completion.

Routines that return pointers return `NULL` on error.

**ATTRIBUTES**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`addch(3CURSES)`, `curses(3CURSES)`, `refresh(3CURSES)`, `subwin(3CURSES)`, `touchline(3CURSES)`, `touchwin(3CURSES)`, `waddch(3CURSES)`, `wnoutrefresh(3CURSES)`, `wrefresh(3CURSES)`, `attributes(5)`

**NOTES**

The header file `<curses.h>` automatically includes the header files `<stdio.h>`, `<unctrl.h>` and `<widec.h>`.

Note that `pechochar()` may be a macro.
curs_printw(3CURSES)

NAME
curs_printw, printw, wprintf, mvprintf, mvwprintf, vwprintf – print formatted output in curses windows

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int printf(char *fmt, /* arg */ ...);
int wprintw(WINDOW *win, char *fmt, /* arg */ ...);
int mvprintf(int y, int x, char *fmt, /* arg */ ...);
int mvwprintw(WINDOW *win, int y, int x, char *fmt, /* arg */...);
#include <varargs.h>
int vwprintw(WINDOW *win, char *fmt, /* varglist */ ...);

DESCRIPTION
The printf(), wprintw(), mvprintf(), and mvwprintw() routines are analogous to printf() (see printf(3C)). In effect, the string that would be output by printf() is output instead as though waddstr() were used on the given window.

The vwprintw() routine is analogous to vprintf() (see vprintf(3C)) and performs a wprintw() using a variable argument list. The third argument is a va_list, a pointer to a list of arguments, as defined in <varargs.h>.

RETURN VALUES
All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), printf(3C), vprintf(3C), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.
curs_refresh(3CURSES)

NAME
curs_refresh, refresh, wrefresh, wnoutrefresh, doupdate, redrawwin, wredrawln –
refresh curses windows and lines

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
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
The refresh() and wrefresh() routines (or wnoutrefresh() and doupdate())
must be called to get any output on the terminal, as other routines merely manipulate
data structures. The routine wrefresh() copies the named window to the physical
terminal screen, taking into account what is already there in order to do optimizations.
The refresh() routine is the same, using stdscr as the default window. Unless
leaveok() has been enabled, the physical cursor of the terminal is left at the location
of the cursor for that window.

The wnoutrefresh() and doupdate() routines allow multiple updates with more
efficiency than wrefresh() alone. In addition to all the window structures, curses
keeps two data structures representing the terminal screen: a physical screen,
describing what is actually on the screen, and a virtual screen, describing what the
programmer wants to have on the screen.

The routine wrefresh() works by first calling wnoutrefresh(), which copies the
named window to the virtual screen, and then calling doupdate(), which compares
the virtual screen to the physical screen and does the actual update. If the programmer
wishes to output several windows at once, a series of calls to wrefresh() results 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() once, resulting in only one burst of output, with fewer
total characters transmitted and less CPU time used. If the win argument to
wrefresh() is the global variable curscr, the screen is immediately cleared and
repainted from scratch.

The redrawwin() routine indicates to curses that some screen lines are corrupted
and should be thrown away before anything is written over them. These routines
could be used for programs such as editors, which want a command to redraw some
part of the screen or the entire screen. The routine redrawln() is preferred over
redrawwin() where a noisy communication line exists and redrawing the entire
window could be subject to even more communication noise. Just redrawing several
lines offers the possibility that they would show up unblemished.
curs_refresh(3CURSES)

RETURN VALUES
All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

curs_outopts(3CURSES), curses(3CURSES), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.

Note that refresh() and redrawwin() may be macros.
curs_scanw(3CURSES)

NAME
curs_scanw, scanw, wscanw, mvscanw, mvwscanw, vwscanw – convert formatted
input from a curses widow

SYNOPSIS
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int scanw(char *fmt, /* arg */ ...);
int wscanw(WINDOW *win, char *fmt, /* arg */ ...);
int mvscanw(int y, int x, char *fmt, /* arg */ ...);
int mvwscanw(WINDOW *win, int y, int x, char *fmt, /* arg */ ...);
int vwscanw(WINDOW *win, char *fmt, va_list vararg);

DESCRIPTION
The scanw(), wscanw(), and mvscanw() routines correspond to scanf() (see
scanf(3C)). The effect of these routines is as though wgetstr() were called on the
window, and the resulting line used as input for the scan. Fields which do not map to
a variable in the fmt field are lost.

The vwscanw() routine is similar to vwprintw() in that it performs a wscanw()
using a variable argument list. The third argument is a va_list, a pointer to a list of
arguments, as defined in <varargs.h>.

RETURN VALUES
vwscanw() returns ERR on failure and an integer equal to the number of fields
scanned on success.

Applications may interrogate the return value from the scanw, wscanw(),
mvscanw(), and mvwscanw() routines to determine the number of fields which were
mapped in the call.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_getstr(3CURSES), curs_printw(3CURSES), curses(3CURSES), scanf(3C),
attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and
<unctrl.h>.

Curses Library Functions 147
curs_scr_dump(3CURSES)

NAME curses_scr_dump, scr_dump, scr_restore, scr_init, scr_set – read (write) a curses screen from (to) a file

SYNOPSIS cc [ flag ... ] file ... -lcurses [ library ... ]  
#include <curses.h>  
int scr_dump(char *filename);  
int scr_restore(char *filename);  
int scr_init(char *filename);  
int scr_set(char *filename);

DESCRIPTION With the scr_dump() routine, the current contents of the virtual screen are written to the file filename.

With the scr_restore() routine, the virtual screen is set to the contents of filename, which must have been written using scr_dump(). The next call to doupdate() restores the screen to the way it looked in the dump file.

With the scr_init() routine, the contents of filename are read in and used to initialize the curses data structures about what the terminal currently has on its screen. If the data is determined to be valid, curses bases its next update of the screen on this information rather than clearing the screen and starting from scratch. scr_init() is used after initscr() or a system(3C) call to share the screen with another process which has done a scr_dump() after its endwin() call. The data is declared invalid if the time-stamp of the tty is old or the terminfo capabilities rmcup() and nrrmc() exist.

The scr_set() routine is a combination of scr_restore() and scr_init(). It tells the program that the information in filename is what is currently on the screen, and also what the program wants on the screen. This can be thought of as a screen inheritance function.

To read (write) a window from (to) a file, use the getwin() and putwin() routines (see curs_util(3CURSES)).

RETURN VALUES All routines return the integer ERR upon failure and OK upon success.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curs_initscr(3CURSES), curs_refresh(3CURSES), curs_util(3CURSES), curses(3CURSES), system(3C), attributes(5)
The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>.

Note that `scr_init()`, `scr_set()`, and `scr_restore()` may be macros.
NAME
curs_scroll, scroll, scrl, wscrl – scroll a curses window

SYNOPSIS
cce [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int scroll(WINDOW *win);
int scrl(int n);
int wscrl(WINDOW *win, int n);

DESCRIPTION
With the scroll() routine, the window is scrolled up one line. This involves moving the lines in the window data structure. As an optimization, if the scrolling region of the window is the entire screen, the physical screen is scrolled at the same time.

With the scrl() and wscrl() routines, for positive n scroll the window up n lines (line i+n becomes i), otherwise scroll the window down n lines. This involves moving the lines in the window character image structure. The current cursor position is not changed.

For these functions to work, scrolling must be enabled via scrollok().

RETURN VALUES
All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_outopts(3CURSES), curses(3CURSES), attributes(5)

NOTES
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.

Note that scrl() and scroll() may be macros.
The `curs_set()` function sets the visibility of the cursor to invisible (0), normal (1), or very visible (2). The exact appearance of normal and very visible cursors is terminal dependent.

**PARAMETERS**

- `visibility` is a value of 0 (invisible), 1 (normal), or 2 (very visible).

**RETURN VALUES**

If the terminal supports the mode specified by the `visibility` parameter, the `curs_set()` function returns the previous cursor state. Otherwise, it returns `ERR`.

**ERRORS**

None.
NAME  
curs, slk_init, slk_set, slk_refresh, slk_noutrefresh, slk_label, slk_clear, slk_restore,
slk_touch, slk_attron, slk_attrset, slk_attroff – curses soft label routines

SYNOPSIS  
cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
int slk_init(int fmt);
int slk_set(int labnum, char *label, int fmt);
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(chtype attrs);
int slk_attrset(chtype attrs);
int slk_attroff(chtype attrs);

DESCRIPTION  
curses manipulates the set of soft function-key labels that exist on many terminals. 
For those terminals that do not have soft labels, curses takes over the bottom line of
stdscr, reducing the size of stdscr and the variable LINES. curses standardizes
on eight labels of up to eight characters each.

To use soft labels, the slk_init() routine must be called before initscr() or
newterm() is called. If initscr() eventually uses a line from stdscr to emulate
the soft labels, then fmt determines how the labels are arranged on the screen. Setting
fmt to 0 indicates a 3-2-3 arrangement of the labels; 1 indicates a 4-4 arrangement.

With the slk_set() routine, labnum is the label number, from 1 to 8. label is the
string to be put on the label, up to eight characters in length. A null string or a null
pointer sets up a blank label. fmt is either 0, 1, or 2, indicating whether the label is to
be left-justified, centered, or right-justified, respectively, within the label.

The slk_refresh() and slk_noutrefresh() routines correspond to the
wrefresh() and wnoutrefresh() routines.

With the slk_label() routine, the current label for label number labnum is returned
with leading and trailing blanks stripped.

With the slk_clear() routine, the soft labels are cleared from the screen.

With the slk_restore() routine, the soft labels are restored to the screen after a
slk_clear() is performed.
With the `slk_touch()` routine, all the soft labels are forced to be output the next time a `slk_noutrefresh()` is performed.

The `slk_attron()`, `slk_attrset()`, and `slk_attroff()` routines correspond to `attron()`, `attrset()`, and `attroff()`. They have an effect only if soft labels are simulated on the bottom line of the screen.

**RETURN VALUES**
Routines that return an integer return `ERR` upon failure and an integer value other than `ERR` upon successful completion.

`slk_label()` returns `NULL` on error.

**ATTRIBUTES**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
curs_attr(3CURSES), curs_initscr(3CURSES), curs_refresh(3CURSES), curses(3CURSES), attributes(5)

**NOTES**
The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Most applications would use `slk_noutrefresh()` because a `wrefresh()` is likely to follow soon.
curs_termattrs(3CURSES)

NAME
curs_termattrs, baudrate, erasechar, has_ic, has_il, killchar, longname, termattrs,
terminame – curses environment query routines

SYNOPSIS
cce [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

int baudrate(void);
char erasechar(void);
int has_ic(void);
int has_il(void);
char killchar(void);
char *longname(void);
ctype termattrs(void);
char *termname(void);

DESCRIPTION

The baudrate() routine returns the output speed of the terminal. The number returned is in bits per second, for example 9600, and is an integer.

With the erasechar() routine, the user’s current erase character is returned.

The has_ic() routine is true if the terminal has insert- and delete-character capabilities.

The has_il() routine is true if the terminal has insert- and delete-line capabilities, or can simulate them using scrolling regions. This might be used to determine if it would be appropriate to turn on physical scrolling using scrollok().

With the killchar() routine, the user’s current line kill character is returned.

The longname() routine returns a pointer to a static area containing a verbose description of the current terminal. The maximum length of a verbose description is 128 characters. It is defined only after the call to initscr() or newterm(). The area is overwritten by each call to newterm() and is not restored by set_term(), so the value should be saved between calls to newterm() if longname() is going to be used with multiple terminals.

If a given terminal doesn’t support a video attribute that an application program is trying to use, curses may substitute a different video attribute for it. The termattrs() function returns a logical OR of all video attributes supported by the terminal. This information is useful when a curses program needs complete control over the appearance of the screen.

The termname() routine returns the value of the environment variable TERM (truncated to 14 characters).

RETURN VALUES

longname() and termname() return NULL on error.
Routines that return an integer return **ERR** upon failure and an integer value other than **ERR** upon successful completion.

**ATTRIBUTES**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

curs_initscr(3CURSES), curs_outopts(3CURSES), curses(3CURSES), attributes(5)

**NOTES**

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>.

Note that `termattrs()` may be a macro.
NAME
curs_termcap, tgetent, tgetflag, tgetnum, tgetstr, tgoto, tputs – curses interfaces (emulated) to the termcap library

SYNOPSIS
curses interfaces to the termcap library

cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>
#include <term.h>

int tgetent(char *bp, char *name);
int tgetflag(char id[2]);
int tgetnum(char id[2]);
char *tgetstr(char id[2], char **area);
char *tgoto(char *cap, int col, int row);
int tputs(char *str, int affcnt, int (*putc)(void));

DESCRIPTION
These routines are included as a conversion aid for programs that use the termcap library. Their parameters are the same and the routines are emulated using the terminfo database. These routines are supported at Level 2 and should not be used in new applications.

The tgetent() routine looks up the termcap entry for name. The emulation ignores the buffer pointer bp.

The tgetflag() routine gets the boolean entry for id.

The tgetnum() routine gets the numeric entry for id.

The tgetstr() routine returns the string entry for id. Use tputs() to output the returned string.

The tgoto() routine instantiates the parameters into the given capability. The output from this routine is to be passed to tputs().

The tputs() routine is described on the curs_terminfo(3CURSES) manual page.

RETURN VALUES
Routines that return an integer return ERR upon failure and an integer value other than ERR upon successful completion.

Routines that return pointers return NULL on error.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curs_termcap(3CURSES), curses(3CURSES), putc(3C), attributes(5)
The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`. 
These low-level routines must be called by programs that have to deal directly with the terminfo database to handle certain terminal capabilities, such as programming function keys. For all other functionality, curses routines are more suitable and their use is recommended.

Initially, setupterm() should be called. Note that setupterm() is automatically called by initscr() and newterm(). This defines the set of terminal-dependent variables (listed in terminfo(4)). The terminfo variables lines and columns are initialized by setupterm() as follows: If use_env(FALSE) has been called, values for lines and columns specified in terminfo are used. Otherwise, if the environment variables LINES and COLUMNS exist, their values are used. If these environment variables do not exist and the program is running in a window, the current window size is used. Otherwise, if the environment variables do not exist, the values for lines and columns specified in the terminfo database are used.

The headers <curses.h> and <term.h> should be included (in this order) to get the definitions for these strings, numbers, and flags. Parameterized strings should be
passed through \texttt{tparm()} to instantiate them. All \textit{terminfo} strings (including the output of \texttt{tparm()}) should be printed with \texttt{tput()} or \texttt{putp()}. Call the \texttt{reset\_shell\_mode()} routine to restore the tty modes before exiting (see \texttt{curs\_kernel(3CURSES)}). Programs which use cursor addressing should output \texttt{enter\_ca\_mode} upon startup and should output \texttt{exit\_ca\_mode} before exiting. Programs desiring shell escapes should call \texttt{reset\_shell\_mode} and output \texttt{exit\_ca\_mode} before the shell is called and should output \texttt{enter\_ca\_mode} and call \texttt{reset\_prog\_mode} after returning from the shell.

The \texttt{setupterm()} routine reads in the \textit{terminfo} database, initializing the \textit{terminfo} structures, but does not set up the output virtualization structures used by \texttt{curses}. The terminal type is the character string \texttt{term}; if \texttt{term} is null, the environment variable \texttt{TERM} is used. All output is to file descriptor \texttt{fdes} which is initialized for output. If \texttt{errret} is not null, then \texttt{setupterm()} returns \texttt{OK} or \texttt{ERR} and stores a status value in the integer pointed to by \texttt{errret}. A status of 1 in \texttt{errret} is normal, 0 means that the terminal could not be found, and -1 means that the \textit{terminfo} database could not be found. If \texttt{errret} is null, \texttt{setupterm()} prints an error message upon finding an error and exits. Thus, the simplest call is:

\begin{verbatim}
setupterm((char *)0, 1, (int *)0);
\end{verbatim}

which uses all the defaults and sends the output to \texttt{stdout}.

The \texttt{setterm()} routine is being replaced by \texttt{setupterm()}. The call:

\begin{verbatim}
setupterm(term, 1, (int *)0)
\end{verbatim}

provides the same functionality as \texttt{setterm(term)}. The \texttt{setterm()} routine is included here for compatibility and is supported at Level 2.

The \texttt{set\_curterm()} routine sets the variable \texttt{cur\_term} to \texttt{nterm}, and makes all of the \textit{terminfo} boolean, numeric, and string variables use the values from \texttt{nterm}.

The \texttt{del\_curterm()} routine frees the space pointed to by \texttt{oterm} and makes it available for further use. If \texttt{oterm} is the same as \texttt{cur\_term}, references to any of the \textit{terminfo} boolean, numeric, and string variables thereafter may refer to invalid memory locations until another \texttt{setupterm()} has been called.

The \texttt{restartterm()} routine is similar to \texttt{setupterm()} and \texttt{initscr()}, except that it is called after restoring memory to a previous state. It assumes that the windows and the input and output options are the same as when memory was saved, but the terminal type and baud rate may be different.

The \texttt{tparm()} routine instantiates the string \texttt{str} with parameters \texttt{pi}. A pointer is returned to the result of \texttt{str} with the parameters applied.

The \texttt{tputs()} routine applies padding information to the string \texttt{str} and outputs it. The \texttt{str} must be a \textit{terminfo} string variable or the return value from \texttt{tparm()}, \texttt{tgetstr()}, or \texttt{tgoto()}. \texttt{affcnt} is the number of lines affected, or 1 if not applicable. \texttt{putc} is a \texttt{putchar()}-like routine to which the characters are passed, one at a time.
The `putp()` routine calls `tputs(str, 1, putchar)`. Note that the output of `putpA()` always goes to `stdout`, not to the `filedes` specified in `setupterm()`.

The `vidputs()` routine displays the string on the terminal in the video attribute mode `attrs`, which is any combination of the attributes listed in `curses(3CURSES)`. The characters are passed to the putchar()-like routine `putc()`.

The `vidattr()` routine is like the `vidputs()` routine, except that it outputs through `putchar()`.

The `mvcur()` routine provides low-level cursor motion.

The `tigetflag()`, `tigetnum()` and `tigetstr()` routines return the value of the capability corresponding to the `terminfo capname` passed to them, such as `xenl`.

With the `tigetflag()` routine, the value −1 is returned if `capname` is not a boolean capability.

With the `tigetnum()` routine, the value −2 is returned if `capname` is not a numeric capability.

With the `tigetstr()` routine, the value (char *)−1 is returned if `capname` is not a string capability.

The `capname` for each capability is given in the table column entitled `capname` code in the capabilities section of `terminfo(4)`. 

```c
char *boolnames, *boolcodes, *boolfnames
char *numnames, *numcodes, *numfnames
char *strnames, *strcodes, *strfnames
```

These null-terminated arrays contain the `capnames`, the `termcap` codes, and the full C names, for each of the `terminfo` variables.

**RETURN VALUES**

All routines return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion, unless otherwise noted in the preceding routine descriptions.

Routines that return pointers always return `NULL` on error.

**ATTRIBUTES**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
curs_initscr(3CURSES), curs_kernel(3CURSES), curs_termcap(3CURSES), curses(3CURSES), putc(3C), terminfo(4), attributes(5)
The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>.

The `setupterm()` routine should be used in place of `setterm()`.

Note that `vidattr()` and `vidputs()` may be macros.
curs_touch(3CURSES)

NAME
curs_touch, touchwin, touchline, untouchwin, wtouchln, is_linetouched,
   is_wintouched – curses refresh control routines

SYNOPSIS
c curses [ flag ... ] file ... -lcurses [ library ... ]
# include <curses.h>

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);
int is_linetouched(WINDOW *win, int line);
int is_wintouched(WINDOW *win);

DESCRIPTION
The touchwin() and touchline() routines throw away all optimization
information about which parts of the window have been touched, by pretending that
the entire window has been drawn on. This is sometimes necessary when using
overlapping windows, since a change to one window affects the other window, but the
records of which lines have been changed in the other window do not reflect the
change. The routine touchline() only pretends that count lines have been changed,
beginning with line start.

The untouchwin() routine marks all lines in the window as unchanged since the last
call to wrefresh().

The wtouchln() routine 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().

The is_linetouched() and is_wintouched() routines return TRUE if the
specified line/window was modified since the last call to wrefresh(); otherwise
they return FALSE. In addition, is_linetouched() returns ERR if line is not valid
for the given window.

RETURN VALUES
All routines return the integer ERR upon failure and an integer value other than ERR
upon successful completion, unless otherwise noted in the preceding routine
descriptions.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curs_refresh(3CURSES), curses(3CURSES), attributes(5)
The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>.

Note that all routines except `wtouchln()` may be macros.
curs_util(3CURSES)

NAME    curs_util, unctrl, keyname, filter, use_env, putwin, getwin, delay_output, flushinp –
curses miscellaneous utility routines

SYNOPSIS    cc [ flag ... ] file ... -lcurses [ library ... ]
#include <curses.h>

char *unctrl(chtype c);
char *keyname(int c);
int filter(void);
void use_env(char bool);
int putwin(WINDOW *win, FILE *filep);
WINDOW *getwin(FILE *filep);
int delay_output(int ms);
int flushinp(void);

DESCRIPTION    The unctrl() macro expands to a character string which is a printable representation
of the character c. Control characters are displayed in the ^X notation. Printing
characters are displayed as is.

With the keyname() routine, a character string corresponding to the key c is returned.

The filter() routine, if used, is called before initscr() or newterm() are called. It makes
curses think that there is a one-line screen. curses does not use any
terminal capabilities that assume that they know on what line of the screen the cursor
is positioned.

The use_env() routine, if used, is called before initscr() or newterm() are
called. When called with FALSE as an argument, the values of lines and columns
specified in the terminfo database will be used, even if environment variables LINES
and COLUMNS (used by default) are set, or if curses is running in a window (in which
case default behavior would be to use the window size if LINES and COLUMNS are not
set).

With the putwin() routine, all data associated with window win is written into the
file to which filep points. This information can be later retrieved using the getwin() function.

The getwin() routine reads window related data stored in the file by putwin(). The
routine then creates and initializes a new window using that data. It returns a pointer
to the new window.

The delay_output() routine inserts an ms millisecond pause in output. This routine
should not be used extensively because padding characters are used rather than a
CPU pause.
The `flushinp()` routine throws away any typeahead that has been typed by the user and has not yet been read by the program.

**RETURN VALUES**
Except for `flushinp()`, routines that return an integer return `ERR` upon failure and an integer value other than `ERR` upon successful completion.

`flushinp()` always returns `OK`.

Routines that return pointers return `NULL` on error.

**ATTRIBUTES**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
`curs_initscr(3CURSES), curs_scr_dump(3CURSES), curses(3CURSES), attributes(5)`

**NOTES**
The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that `unctrl()` is a macro, which is defined in `<unctrl.h>`.
The `newwin()` routine creates and returns a pointer to a new window with the given number of lines, `nlines`, and columns, `ncols`. The upper left-hand corner of the window is at line `begin_y`, column `begin_x`. If either `nlines` or `ncols` is zero, they default to `LINES` — `begin_y` and `COLS` — `begin_x`. A new full-screen window is created by calling `newwin(0,0,0,0)`.

The `delwin()` routine deletes the named window, freeing all memory associated with it. Subwindows must be deleted before the main window can be deleted.

The `mvwin()` routine moves the window so that the upper left-hand corner is at position `(x,y)`. 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.

The `subwin()` routine creates and returns a pointer to a new window with the given number of lines, `nlines`, and columns, `ncols`. The window is at position `(begin_y, begin_x)` on the screen. (This position is relative to the screen, and not to the window `orig`.) The window is made in the middle of the window `orig`, so that changes made to one window will affect both windows. The subwindow shares memory with the window `orig`. When using this routine, it is necessary to call `touchwin()` or `touchline()` on `orig` before calling `wrefresh()` on the subwindow.
The `derwin()` routine is the same as `subwin()`, except that `begin_y` and `begin_x` are relative to the origin of the window `orig` rather than the screen. There is no difference between the subwindows and the derived windows.

The `mvderwin()` routine 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.

The `dupwin()` routine creates an exact duplicate of the window `win`.

Each `curses` window maintains two data structures: the character image structure and the status structure. The character image structure is shared among all windows in the window hierarchy (that is, the window with all subwindows). The status structure, which contains information about individual line changes in the window, is private to each window. The routine `wrefresh()` uses the status data structure when performing screen updating. Since status structures are not shared, changes made to one window in the hierarchy may not be properly reflected on the screen.

The routine `wsyncup()` causes the changes in the status structure of a window to be reflected in the status structures of its ancestors. If `syncok()` is called with second argument `TRUE` then `wsyncup()` is called automatically whenever there is a change in the window.

The routine `wcursyncup()` updates the current cursor position of all the ancestors of the window to reflect the current cursor position of the window.

The routine `wsyncdown()` updates the status structure of the window to reflect the changes in the status structures of its ancestors. Applications seldom call this routine because it is called automatically by `wrefresh()`.

**RETURN VALUES**

Routines that return an integer return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion.

`delwin()` returns the integer `ERR` upon failure and `OK` upon successful completion.

Routines that return pointers return `NULL` on error.

**ATTRIBUTES**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

curs_refresh(3CURSES), curs_touch(3CURSES), curses(3CURSES), attributes(5)
The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h>.

If many small changes are made to the window, the wsyncup() option could degrade performance.

Note that syncok() may be a macro.
NAME  
cur_term – current terminal information

SYNOPSIS  
#include <curses.h>

extern TERMINAL *cur_term;

DESCRIPTION  
The external variable cur_term identifies the record in the terinfo associated with the terminal currently in use.

SEE ALSO  
set_curterm(3XCURSES), tigetflag(3XCURSES)
def_prog_mode(3XCURSES)

NAME
    def_prog_mode, def_shell_mode, reset_prog_mode, reset_shell_mode – save/restore terminal modes

SYNOPSIS
    #include <curses.h>

    int def_prog_mode(void);
    int def_shell_mode(void);
    int reset_prog_mode(void);
    int reset_shell_mode(void);

DESCRIPTION
    The def_prog_mode() and def_shell_mode() functions save the current terminal modes as "program" (within X/Open Curses) or "shell" (outside X/Open Curses). The modes are saved automatically by initscr(3XCURSES), newterm(3XCURSES), and setupterm(3XCURSES).

    The reset_prog_mode() and reset_shell_mode() functions reset the current terminal modes to "program" (within X/Open Curses) or "shell" (outside X/Open Curses). The endwin(3XCURSES) function automatically calls the reset_shell_mode() function and the doupdate(3XCURSES) function calls the reset_prog_mode() function after calling endwin().

RETURN VALUES
    On success, these functions return OK. Otherwise, they return ERR.

ERRORS
    None.

SEE ALSO
    endwin(3XCURSES), initscr(3XCURSES), newterm(3XCURSES), setupterm(3XCURSES)
### NAME
delay_output – delays output

### SYNOPSIS
```
#include <curses.h>

int delay_output(int ms);
```

### DESCRIPTION
The `delay_output` function delays output for `ms` milliseconds by inserting pad characters in the output stream.

### PARAMETERS
- **ms**: Is the number of milliseconds to delay the output.

### RETURN VALUES
On success, the `delay_output` function returns `OK`. Otherwise, it returns `ERR`.

### ERRORS
None.

### SEE ALSO
napms(3XCURSES)
delch(3XCURSES)

NAME  | delch, mvdelch, mvwdelch, wdelch – remove a character

SYNOPSIS | #include <curses.h>

int delch(void);
int mvdelch(int y, int x);
int mvwdelch(WINDOW *win, int y, int x);
int wdelch(WINDOW *win);

DESCRIPTION | The delch() and wdelch() functions delete the character at the current cursor position from stdscr and win, respectively. All remaining characters after cursor through to the end of the line are shifted one character towards the start of the line. The last character on the line becomes a space; characters on other lines are not affected.

The mvdelch() and mvwdelch() functions delete the character at the position specified by the x and y parameters; the former deletes the character from stdscr; the latter from win.

PARAMETERS | y  | Is the y (row) coordinate of the position of the character to be removed.
          | x  | Is the x (column) coordinate of the position of the character to be removed.
          | win| Is a pointer to the window containing the character to be removed.

RETURN VALUES | On success, these functions return OK. Otherwise, they return ERR.

ERRORS | None.

SEE ALSO | bkgdset(3XCURSES), insch(3XCURSES)
del_curterm, restartterm, set_curterm, setterm, setupterm – free space pointed to by terminal

#include <term.h>

int del_curterm (TERMINAL *oterm);
int restartterm (char *term, int fdlen, int *errret);
TERMINAL *set_curterm (TERMINAL *nterm);
int setterm (char *term);
int setupterm (char *term, int fdlen, int *errret);

Within X/Open Curses, the setupterm() function is automatically called by the initscr (3XC) and newterm (3XC) functions. This function can be also be used outside of X/Open Curses when a program has to deal directly with the terminfo database to handle certain terminal capabilities. The use of appropriate X/Open Curses functions is recommended in all other situations.

The setupterm() function loads terminal-dependent variables for the terminfo layer of X/Open Curses. The setupterm() function initializes the terminfo variables lines and columns such that if use_env(FALSE) has been called, the terminfo values assigned in the database are used regardless of the environmental variables LINES and COLUMNS or the program’s window dimensions; when use_env(TRUE) has been called, which is the default, the environment variables LINES and COLUMNS are used, if they exist. If the environment variables do not exist and the program is running in a window, the current window size is used.

The term parameter of setupterm() specifies the terminal; if null, terminal type is taken from the TERM environment variable. All output is sent to fdlen which is initialized for output. If errret is not null, OK or ERR is returned and a status value is stored in the integer pointed to by errret. The following status values may be returned:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normal</td>
</tr>
<tr>
<td>0</td>
<td>Terminal could not be found</td>
</tr>
<tr>
<td>-1</td>
<td>terminfo database could not be found</td>
</tr>
</tbody>
</table>

If errret is null, an error message is printed, and the setupterm() function calls the exit() function with a non-zero parameter.

The setterm() macro is an older version of setupterm(). It is included for compatibility with previous versions of Curses. New programs should use setupterm().
The `set_curterm()` function sets the `cur_term` variable to `nterm`. The values from `nterm` as well as other state information for the terminal are used by X/Open Curses functions such as `beep(3XCURSES)`, `flash(3XCURSES)`, `mvcur(3XCURSES)`, `tigetflag(3XCURSES)`, `tigetstr(3XCURSES)`, and `tigetnum(3XCURSES)`.

The `del_curterm()` function frees the space pointed to by `oterm`. If `oterm` and the `cur_term` variable are the same, all Boolean, numeric, or string `terminfo` variables will refer to invalid memory locations until you call `setupterm()` and specify a new terminal type.

The `restartterm()` function assumes that a call to `setupterm()` has already been made (probably from `initscr()` or `newterm()`). It allows you to specify a new terminal type in `term` and updates the data returned by `baudrate(3XCURSES)` based on `fildes`. Other information created by the `initscr()`, `newterm()`, and `setupterm()` functions is preserved.

**PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>oterm</code></td>
<td>Is the terminal type for which to free space.</td>
</tr>
<tr>
<td><code>term</code></td>
<td>Is the terminal type for which variables are set.</td>
</tr>
<tr>
<td><code>fildes</code></td>
<td>Is a file descriptor initialized for output.</td>
</tr>
<tr>
<td><code>errret</code></td>
<td>Is a pointer to an integer in which the status value is stored.</td>
</tr>
<tr>
<td><code>nterm</code></td>
<td>Is the new terminal to become the current terminal.</td>
</tr>
</tbody>
</table>

**RETURN VALUES**

On success, the `set_curterm()` function returns the previous value of `cur_term`. Otherwise, it returns a null pointer.

On success, the other functions return `OK`. Otherwise, they return `ERR`.

**ERRORS**

None.

**SEE ALSO**

`baudrate(3XCURSES)`, `beep(3XCURSES)`, `initscr(3XCURSES)`, `mvcur(3XCURSES)`, `tigetflag(3XCURSES)`, `use_env(3XCURSES)`
deleteln(3XCURSES)

NAME
deleeln, wdeleteln – remove a line

SYNOPSIS
#include <curses.h>

int deleteln(void);
int wdeleteln(WINDOW *win);

DESCRIPTION
The deleteln() and wdeleteln() functions delete the line containing the cursor
from stdscr and win, respectively. All lines below the one deleted are moved up one
line. The last line of the window becomes blank. The position of the cursor is
unchanged.

PARAMETERS
win Is a pointer to the window from which the line is removed.

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None.

SEE ALSO
bkgdset(3XCURSES), insdelln(3XCURSES), insertln(3XCURSES)
delscreen(3XCURSES)

<table>
<thead>
<tr>
<th>NAME</th>
<th>delscreen – free space associated with the SCREEN data structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>#include &lt;curses.h&gt;</td>
</tr>
<tr>
<td></td>
<td>void delscreen(SCREEN *sp);</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The delscreen() function frees space associated with the SCREEN data structure. This function should be called after endwin(3XCURSES) if a SCREEN data structure is no longer needed.</td>
</tr>
<tr>
<td>PARAMETERS</td>
<td>sp Is a pointer to the screen structure for which to free space.</td>
</tr>
<tr>
<td>RETURN VALUES</td>
<td>The delscreen() function does not return a value.</td>
</tr>
<tr>
<td>ERRORS</td>
<td>None.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>endwin(3XCURSES), initscr(3XCURSES), newterm(3XCURSES)</td>
</tr>
</tbody>
</table>
The `delwin()` function deletes the specified window, freeing up the memory associated with it.

Deleting a parent window without deleting its subwindows and then trying to manipulate the subwindows will have undefined results.

**PARAMETERS**

- `win` Is a pointer to the window that is to be deleted.

**RETURN VALUES**

On success, this functions returns `OK`. Otherwise, it returns `ERR`.

**ERRORS**

None.

**SEE ALSO**

`derwin(3XCURSES), dupwin(3XCURSES)`
### NAME

derwin, newwin, subwin – create a new window or subwindow

### SYNOPSIS

```c
#include <curses.h>

WINDOW *derwin(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);

WINDOW *newwin(int nlines, int ncols, int begin_y, int begin_x);

WINDOW *subwin(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);
```

### DESCRIPTION

The `derwin()` function creates a subwindow within window `orig`, with the specified number of lines and columns, and upper left corner positioned at `begin_x`, `begin_y` relative to window `orig`. A pointer to the new window structure is returned.

The `newwin()` function creates a new window with the specified number of lines and columns and upper left corner positioned at `begin_x`, `begin_y`. A pointer to the new window structure is returned. A full-screen window can be created by calling `newwin(0,0,0,0)`.

If the number of lines specified is zero, `newwin()` uses a default value of `LINES` minus `begin_y`; if the number of columns specified is zero, `newwin()` uses the default value of `COLS` minus `begin_x`.

The `subwin()` function creates a subwindow within window `orig`, with the specified number of lines and columns, and upper left corner positioned at `begin_x`, `begin_y` (relative to the physical screen, not to window `orig`). A pointer to the new window structure is returned.

The original window and subwindow share character storage of the overlapping area (each window maintains its own pointers, cursor location, and other items). This means that characters and attributes are identical in overlapping areas regardless of which window characters are written to.

When using subwindows, it is often necessary to call `touchwin(3XCURSES)` before `wrefresh(3XCURSES)` to maintain proper screen contents.

### PARAMETERS

- `orig` Is a pointer to the parent window for the newly created subwindow.
- `nlines` Is the number of lines in the subwindow.
- `ncols` Is the number of columns in the subwindow.
- `begin_y` Is the y (row) coordinate of the upper left corner of the subwindow, relative to the parent window.
- `begin_x` Is the x (column) coordinate of the upper left corner of the subwindow, relative to the parent window.

### RETURN VALUES

On success, these functions return a pointer to the newly-created window. Otherwise, they return `ERR`. 

178 man pages section 3: Curses Library Functions • Last Revised 1 Jun 1996
<table>
<thead>
<tr>
<th>ERRORS</th>
<th>None.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEE ALSO</td>
<td><code>doupdate(3XCURSES)</code>, <code>is_linetouched(3XCURSES)</code></td>
</tr>
</tbody>
</table>
#include <curses.h>

int doupdate(void);
int refresh(void);
int wnoutrefresh(WINDOW *win);
int wrefresh(WINDOW *win);

The refresh() and wrefresh() functions copy stdscr and win, respectively, to
the terminal screen. These functions call the wnoutrefresh() function to copy the
specified window to curscr and the doupdate() function to do the actual update.
The physical cursor is mapped to the same position as the logical cursor of the last
window to update curscr unless leaveok(3XCURSES) is enabled (in which case, the
cursor is placed in a position that X/Open Curses finds convenient).

When outputting several windows at once, it is often more efficient to call the
wnoutrefresh() and doupdate() functions directly. A call to wnoutrefresh()
for each window, followed by only one call to doupdate() to update the screen,
results in one burst of output, fewer characters sent, and less CPU time used.

If the win parameter to wrefresh() is the global variable curscr, the screen is
immediately cleared and repainted from scratch.

For details on how the wnoutrefresh() function handles overlapping windows
with broad glyphs, see the Overlapping Windows section of the
curses(3XCURSES) reference manual page.

win Is a pointer to the window in which to refresh.

On success, these functions return OK. Otherwise, they return ERR.

None.

clearok(3XCURSES), curses(3XCURSES), prefresh(3XCURSES),
redrawwin(3XCURSES)
The `dupwin()` function creates a duplicate of window `win`. A pointer to the new window structure is returned.

### Description

- **Parameters**
  - `win` is a pointer to the window that is to be duplicated.

- **Return Values**
  - On success, this function returns a pointer to new window structure; otherwise, it returns a null pointer.

- **Errors**
  - None.

- **See Also**
  - `delwin(3XCURSES)`, `derwin(3XCURSES)`
echo(3XCURSES)

NAME        echo, noecho – enable/disable terminal echo
SYNOPSIS    #include <curses.h>
             int echo(void);
             int noecho(void);

DESCRIPTION The echo() function enables Echo mode for the current screen. The noecho() function disables Echo mode for the current screen. Initially, curses software echo mode is enabled and hardware echo mode of the tty driver is disabled. The echo() and noecho() functions control software echo only. Hardware echo must remain disabled for the duration of the application, else the behavior is undefined.

RETURN VALUES Upon successful completion, these functions return OK. Otherwise, they return ERR.

ERRORS No errors are defined.

SEE ALSO  getch(3XCURSES), getstr(3XCURSES), initscr(3XCURSES), scanw(3XCURSES)
echochar(3XCURSES)

NAME
  echochar, wechochar – add a single-byte character and refresh window

SYNOPSIS
  #include <curses.h>

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

DESCRIPTION
  The echochar() function produces the same effect as calling addch(3XCURSES) and then refresh(3XCURSES). The wechochar() function produces the same effect as calling waddch(3XCURSES) and then wrefresh(3XCURSES).

PARAMETERS
  ch Is a pointer to the character to be written to the window.
  win Is a pointer to the window in which the character is to be added.

RETURN VALUES
  On success, these functions return OK. Otherwise, they return ERR.

ERRORS
  None.

SEE ALSO
  addch(3XCURSES), doupdate(3XCURSES), echo_wchar(3XCURSES)
echo_wchar(3XCURSES)

NAME  echo_wchar, wecho_wchar – add a complex character and refresh window

SYNOPSIS  
```c
#include <curses.h>

int echo_wchar(const cchar_t *wch);
int wecho_wchar(WINDOW *win, const cchar_t *wch);
```

DESCRIPTION  The `echo_wchar()` function produces the same effect as calling `add_wch(3XCURSES)` and then `refresh(3XCURSES)`. The `wecho_wchar()` function produces the same effect as calling `wadd_wch(3XCURSES)` and then `wrefresh(3XCURSES).

PARAMETERS  
- `wch`  Is a pointer to the complex character to be written to the window.
- `win`  Is a pointer to the window in which the character is to be added.

RETURN VALUES  On success, these functions return `OK`. Otherwise, they return `ERR`.

ERRORS  None.

SEE ALSO  `add_wch(3XCURSES)`, `doupdate(3XCURSES)`, `echochar(3XCURSES)`
endwin(3XCURSES)

<table>
<thead>
<tr>
<th>NAME</th>
<th>endwin, isendwin – restore initial terminal environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>#include &lt;curses.h&gt;</td>
</tr>
<tr>
<td></td>
<td>int endwin(void);</td>
</tr>
<tr>
<td></td>
<td>bool isendwin(void);</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The endwin() function restores the terminal after Curses activity by at least restoring the saved shell terminal mode, flushing any output to the terminal, and moving the cursor to the first column of the last line of the screen. Refreshing a window resumes program mode. The application must call endwin() for each terminal being used before exiting. If newterm(3XCURSES) is called more than once for the same terminal, the first screen created must be the last one for which endwin() is called.</td>
</tr>
<tr>
<td></td>
<td>The isendwin() function indicates whether or not a screen has been refreshed since the last call to endwin().</td>
</tr>
<tr>
<td>RETURN VALUES</td>
<td>Upon successful completion, the endwin() function returns OK. Otherwise, it returns ERR.</td>
</tr>
<tr>
<td></td>
<td>The isendwin() function returns TRUE if endwin() has been called without any subsequent refresh. Otherwise, it returns FALSE.</td>
</tr>
<tr>
<td>ERRORS</td>
<td>Non errors are defined.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>doupdate(3XCURSES), newterm(3XCURSES)</td>
</tr>
</tbody>
</table>
erasechar(3XCURSES)

NAME
erasechar, erasewchar, killchar, killwchar – return current ERASE or KILL characters

SYNOPSIS
#include <curses.h>

char erasechar(void);
int erasewchar(wchar_t *ch);
char killchar(void);
int killwchar(wchar_t *ch);

DESCRIPTION
The erasechar() function returns the current ERASE character from the tty driver. This character is used to delete the previous character during keyboard input. The returned value can be used when including deletion capability in interactive programs.

The killchar() function is similar to erasechar(). It returns the current KILL character.

The erasewchar() and killwchar() functions are similar to erasechar() and killchar() respectively, but store the ERASE or KILL character in the object pointed to by ch.

PARAMETERS
ch Is a pointer to a location where a character may be stored.

RETURN VALUES
For erasechar() and killchar(), the terminal’s current ERASE or KILL character is returned.

On success, the erasewchar() and killwchar() functions return OK. Otherwise, they return ERR.

SEE ALSO
getch(3XCURSES), getstr(3XCURSES), get_wch(3XCURSES)
NAME | filter – disable use of certain terminal capabilities  
SYNOPSIS | #include <curses.h>  
 | void filter(void);  
DESCRIPTION | The filter() function changes how X/Open Curses initializes terminal capabilities that assume the terminal has more than one line. After a call to filter(), the \texttt{initscr(3XCURSES)} or \texttt{newterm(3XCURSES)} functions also:  
 | - Disable use of \texttt{clear}, \texttt{cud}, \texttt{cud1}, \texttt{cup}, \texttt{cuu1}, and \texttt{vpa}.  
 | - Set home string to the value of \texttt{cr}.  
 | - Set lines to 1.  
RETURN VALUES | The filter() function does not return a value.  
ERRORS | None.  
SEE ALSO | \texttt{initscr(3XCURSES)}, \texttt{newterm(3XCURSES)}
flushinp(3XCURSES)

NAME
flushinp – discard type-ahead characters

SYNOPSIS
#include <curses.h>

int flushinp(void);

DESCRIPTION
The flushinp() function discards (flushes) any characters in the input buffer
associated with the current screen.

RETURN VALUES
The flushinp() function always returns OK.

ERRORS
No errors are defined.
NAME  form_cursor, pos_form_cursor - position forms window cursor

SYNOPSIS  
```c
#include <form.h>

int pos_form_cursor(FORM *form);
```

DESCRIPTION  pos_form_cursor() moves the form window cursor to the location required by the form driver to resume form processing. This may be needed after the application calls a curses library I/O routine.

RETURN VALUES  pos_form_cursor() returns one of the following:
- E_OK  The function returned successfully.
- E_SYSTEM_ERROR  System error.
- E_BAD_ARGUMENT  An argument is incorrect.
- E_NOT_POSTED  The form is not posted.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3CURSES), forms(3CURSES), attributes(5)

NOTES  The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.
form_data(3CURSES)

NAME    form_data, data_ahead, data_behind – tell if forms field has off-screen data ahead or behind

SYNOPSIS  cc [ flag ... ] file ... -lform -lcurses [ library .. ]
# include <form.h>

int data_ahead(FORM *form);
int data_behind(FORM *form);

DESCRIPTION  data_ahead() returns TRUE (1) if the current field has more off-screen data ahead; otherwise it returns FALSE (0).

data_behind() returns TRUE (1) if the current field has more off-screen data behind; otherwise it returns FALSE (0).

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3CURSES), forms(3CURSES), attributes(5)

NOTES  The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME
form_driver – command processor for the forms subsystem

SYNOPSIS
cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>

int form_driver(FORM *form, int c);

DESCRIPTION
form_driver() is the workhorse of the forms subsystem; it checks to determine whether the character c is a forms request or data. If it is a request, the form driver executes the request and reports the result. If it is data (a printable ASCII character), it enters the data into the current position in the current field. If it is not recognized, the form driver assumes it is an application-defined command and returns E_UNKNOWN_COMMAND. Application defined commands should be defined relative to MAX_COMMAND, the maximum value of a request listed below.

Form driver requests:
- REQ_NEXT_PAGE  Move to the next page.
- REQ_PREV_PAGE  Move to the previous page.
- REQ_FIRST_PAGE Move to the first page.
- REQ_LAST_PAGE  Move to the last page.
- REQ_NEXT_FIELD Move to the next field.
- REQ_PREV_FIELD Move to the previous field.
- REQ_FIRST_FIELD Move to the first field.
- REQ_LAST_FIELD  Move to the last field.
- REQ_SNEXT_FIELD Move to the sorted next field.
- REQ_SPREV_FIELD Move to the sorted prev field.
- REQ_SFIRST_FIELD Move to the sorted first field.
- REQ_SLAST_FIELD Move to the sorted last field.
- REQ_LEFT_FIELD  Move left to field.
- REQ_RIGHT_FIELD Move right to field.
- REQ_UP_FIELD    Move up to field.
- REQ_DOWN_FIELD  Move down to field.
- REQ_NEXT_CHAR   Move to the next character in the field.
- REQ_PREV_CHAR   Move to the previous character in the field.
- REQ_NEXT_LINE   Move to the next line in the field.
- REQ_PREV_LINE   Move to the previous line in the field.
- REQ_NEXT_WORD   Move to the next word in the field.
form_driver(3CURSES)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ_PREV_WORD</td>
<td>Move to the previous word in the field.</td>
</tr>
<tr>
<td>REQ_BEG_FIELD</td>
<td>Move to the first char in the field.</td>
</tr>
<tr>
<td>REQ_END_FIELD</td>
<td>Move after the last char in the field.</td>
</tr>
<tr>
<td>REQ_BEG_LINE</td>
<td>Move to the beginning of the line.</td>
</tr>
<tr>
<td>REQ_END_LINE</td>
<td>Move after the last char in the line.</td>
</tr>
<tr>
<td>REQ_LEFT_CHAR</td>
<td>Move left in the field.</td>
</tr>
<tr>
<td>REQ_RIGHT_CHAR</td>
<td>Move right in the field.</td>
</tr>
<tr>
<td>REQ_UP_CHAR</td>
<td>Move up in the field.</td>
</tr>
<tr>
<td>REQ_DOWN_CHAR</td>
<td>Move down in the field.</td>
</tr>
<tr>
<td>REQ_NEW_LINE</td>
<td>Insert/overlay a new line.</td>
</tr>
<tr>
<td>REQ_INS_CHAR</td>
<td>Insert the blank character at the cursor.</td>
</tr>
<tr>
<td>REQ_INS_LINE</td>
<td>Insert a blank line at the cursor.</td>
</tr>
<tr>
<td>REQ_DEL_CHAR</td>
<td>Delete the character at the cursor.</td>
</tr>
<tr>
<td>REQ_DEL_PREV</td>
<td>Delete the character before the cursor.</td>
</tr>
<tr>
<td>REQ_DEL_LINE</td>
<td>Delete the line at the cursor.</td>
</tr>
<tr>
<td>REQ_DEL_WORD</td>
<td>Delete the word at the cursor.</td>
</tr>
<tr>
<td>REQ_CLR_EOL</td>
<td>Clear to the end of the line.</td>
</tr>
<tr>
<td>REQ_CLR_EOF</td>
<td>Clear to the end of the field.</td>
</tr>
<tr>
<td>REQ_CLR_FIELD</td>
<td>Clear the entire field.</td>
</tr>
<tr>
<td>REQ_OVL_MODE</td>
<td>Enter overlay mode.</td>
</tr>
<tr>
<td>REQ_INS_MODE</td>
<td>Enter insert mode.</td>
</tr>
<tr>
<td>REQ_SCR_FLINE</td>
<td>Scroll the field forward a line.</td>
</tr>
<tr>
<td>REQ_SCR_BLINE</td>
<td>Scroll the field backward a line.</td>
</tr>
<tr>
<td>REQ_SCR_FPAGE</td>
<td>Scroll the field forward a page.</td>
</tr>
<tr>
<td>REQ_SCR_BPAGE</td>
<td>Scroll the field backward a page.</td>
</tr>
<tr>
<td>REQ_SCR_FHPAGE</td>
<td>Scroll the field forward half a page.</td>
</tr>
<tr>
<td>REQ_SCR_BHPAGE</td>
<td>Scroll the field backward half a page.</td>
</tr>
<tr>
<td>REQ_SCR_FCHAR</td>
<td>Horizontal scroll forward a character.</td>
</tr>
<tr>
<td>REQ_SCR_BCHAR</td>
<td>Horizontal scroll backward a character.</td>
</tr>
<tr>
<td>REQ_SCR_HFLINE</td>
<td>Horizontal scroll forward a line.</td>
</tr>
<tr>
<td>REQ_SCR_HBLINE</td>
<td>Horizontal scroll backward a line.</td>
</tr>
</tbody>
</table>
form_driver(3CURSES)

REQ_SCR_HFHALF  Horizontal scroll forward half a line.
REQ_SCR_HBHALF  Horizontal scroll backward half a line.
REQ_VALIDATION  Validate field.
REQ_PREV_CHOICE Display the previous field choice.
REQ_NEXT_CHOICE Display the next field choice.

RETURN VALUES
form_driver() returns one of the following:
F_OK The function returned successfully.
F_SYSTEM_ERROR System error.
F_BAD_ARGUMENT An argument is incorrect.
F_NOT_POSTED The form is not posted.
F_INVALID_FIELD The field contents are invalid.
F_BAD_STATE The routine was called from an initialization or termination function.
F_REQUEST_DENIED The form driver request failed.
F_UNKNOWN_COMMAND An unknown request was passed to the form driver.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
  curses(3CURSES), forms(3CURSES), attributes(5)

NOTES
The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME
form_field, set_form_fields, form_fields, field_count, move_field – connect fields to forms

SYNOPSIS
cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>

int set_form_fields(FORM *form, FIELD **field);
FIELD **form_fields(FORM *form);
int field_count(FORM *form);
int move_field(FIELD *field, int frow, int fcol);

DESCRIPTION
set_form_fields() changes the fields connected to form to fields. The original fields are disconnected.
form_fields() returns a pointer to the field pointer array connected to form.
field_count() returns the number of fields connected to form.
move_field() moves the disconnected field to the location frow, fcol in the forms subwindow.

RETURN VALUES
form_fields() returns NULL on error.
field_count() returns -1 on error.
set_form_fields() and move_field() return one of the following:
E_OK The function returned successfully.
E_CONNECTED The field is already connected to a form.
E_SYSTEM_ERROR System error.
E_BAD_ARGUMENT An argument is incorrect
E_POSTED The form is posted.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), forms(3CURSES), attributes(5)

NOTES
The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
form_field_attributes(3CURSES)

NAME
form_field_attributes, set_field_fore, field_fore, set_field_back, field_back, set_field_pad, field_pad – format the general display attributes of forms

SYNOPSIS
cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>

int set_field_fore(FIELD *field, chtype attr);
chtype field_fore(FIELD *field);
int set_field_back(FIELD *field, chtype attr);
chtype field_back(FIELD *field);
int set_field_pad(FIELD *field, int pad);
int field_pad(FIELD *field);

DESCRIPTION
set_field_fore() sets the foreground attribute of field. The foreground attribute is the low-level curses display attribute used to display the field contents. field_fore() returns the foreground attribute of field.

set_field_back() sets the background attribute of field. The background attribute is the low-level curses display attribute used to display the extent of the field. field_back() returns the background attribute of field.

set_field_pad() sets the pad character of field to pad. The pad character is the character used to fill within the field. field_pad() returns the pad character of field.

RETURN VALUES
field_fore(), field_back(), and field_pad() return default values if field is NULL. If field is not NULL and is not a valid FIELD pointer, the return value from these routines is undefined.

set_field_fore(), set_field_back(), and set_field_pad() return one of the following:
F_OK The function returned successfully.
F_SYSTEM_ERROR System error.
F_BAD_ARGUMENT An argument is incorrect.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), forms(3CURSES), attributes(5)

NOTES
The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
form_field_buffer(3CURSES)

NAME
form_field_buffer, set_field_buffer, field_buffer, set_field_status, field_status, set_max_field – set and get forms field attributes

SYNOPSIS
ce [ flag ... ] file ... -lform -lcurses [ library ..]
#include <form.h>

int set_field_buffer(FIELD *field, int buf, char *value);
char *field_buffer(FIELD *field, int buf);
int set_field_status(FIELD *field, int status);
int field_status(FIELD *field);
int set_max_field(FIELD *field, int max);

DESCRIPTION
set_field_buffer() sets buffer buf of field to value. Buffer 0 stores the displayed contents of the field. Buffers other than 0 are application specific and not used by the forms library routines. field_buffer() returns the value of field buffer buf.

Every field has an associated status flag that is set whenever the contents of field buffer 0 changes. set_field_status() sets the status flag of field to status. field_status() returns the status of field.

set_max_field() sets a maximum growth on a dynamic field, or if max=0 turns off any maximum growth.

RETURN VALUES
field_buffer() returns NULL on error.
field_status() returns TRUE or FALSE.

set_field_buffer(), set_field_status(), and set_max_field() return one of the following:
E_OK The function returned successfully.
E_SYSTEM_ERROR System error
E_BAD_ARGUMENT An argument is incorrect.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), forms(3CURSES), attributes(5)

NOTES
The header <form.h> automatically includes the headers <eti.h> and <curses.h>.

196 man pages section 3: Curses Library Functions • Last Revised 31 Dec 1996
form_field_info(3CURSES)

NAME  form_field_info, field_info, dynamic_field_info – get forms field characteristics

SYNOPSIS  
```
cc [ flag ... ] file ... -lform -lcurses [ library ... ] 
#include <form.h>
```

```c
int field_info(FIELD *field, int *rows, int *cols, int *frow, int *fcol, 
int *nrow, int *nbuf);
```

```c
int dynamic_field_info(FIELD *field, int *drows, int *dcols, int *
max);
```

DESCRIPTION  
field_info() returns the size, position, and other named field characteristics, as defined in the original call to new_field(), to the locations pointed to by the arguments rows, cols, frow, fcol, nrow, and nbuf.

dynamic_field_info() returns the actual size of the field in the pointer arguments drows, dcols and returns the maximum growth allowed for field in max. If no maximum growth limit is specified for field, max will contain 0. A field can be made dynamic by turning off the field option O_STATIC.

RETURN VALUES  
These routines return one of the following:

E_OK The function returned successfully.
E_SYSTEM_ERROR System error.
E_BAD_ARGUMENT An argument is incorrect.

ATTRIBUTES  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curses(3CURSES), forms(3CURSES), attributes(5)

NOTES  
The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME
form_field_just, set_field_just, field_just – format the general appearance of forms

SYNOPSIS
cc [ flag ...] file ... -lform -lcurses [ library ...]
#include <form.h>

int set_field_just(FIELD *field, int justification);
int field_just(FIELD *field);

DESCRIPTION
set_field_just() sets the justification for field. Justification may be one of:

NO_JUSTIFICATION
JUSTIFY_RIGHT
JUSTIFY_LEFT
JUSTIFY_CENTER

The field justification will be ignored if field is a dynamic field.

field_just() returns the type of justification assigned to field.

RETURN VALUES
field_just() returns one of the following:

NO_JUSTIFICATION
JUSTIFY_RIGHT
JUSTIFY_LEFT
JUSTIFY_CENTER.

set_field_just() returns one of the following:

E_OK The function returned successfully.
E_SYSTEM_ERROR System error.
E_BAD_ARGUMENT An argument is incorrect.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), forms(3CURSES), attributes(5)

NOTES
The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
form_field_new(3CURSES)

NAME
form_field_new, new_field, dup_field, link_field, free_field – create and destroy forms fields

SYNOPSIS
cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>
FIELD *new_field(int r, int c, int frow, int fcol, int nrow, int ncol);
FIELD *dup_field(FIELD *field, int frow, int fcol);
FIELD *link_field(FIELD *field, int frow, int fcol);
int free_field(FIELD *field);

DESCRIPTION
new_field() creates a new field with r rows and c columns, starting at frow, fcol, in
the subwindow of a form. nrow is the number of off-screen rows and nbuf is the
number of additional working buffers. This routine returns a pointer to the new field.

dup_field() duplicates field at the specified location. All field attributes are
duplicated, including the current contents of the field buffers.

link_field() also duplicates field at the specified location. However, unlike
dup_field(), the new field shares the field buffers with the original field. After
creation, the attributes of the new field can be changed without affecting the original
field.

free_field() frees the storage allocated for field.

RETURN VALUES
Routines that return pointers return NULL on error. free_field() returns one of the
following:
E_OK The function returned successfully.
E_CONNECTED The field is already connected to a form.
E_SYSTEM_ERROR System error.
E_BAD_ARGUMENT An argument is incorrect.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), forms(3CURSES), attributes(5)

NOTES
The header <form.h> automatically includes the headers <eti.h> and
<curses.h>.

Curses Library Functions 199
form_field_opts(3CURSES)

NAME     form_field_opts, set_field_opts, field_opts_on, field_opts_off, field_opts – forms field option routines

SYNOPSIS  cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>

int set_field_opts(FIELD *field, OPTIONS opts);
int set_field_opts(FIELD *field, OPTIONS opts);
int field_opts_on(FIELD *field, OPTIONS opts);
int field_opts_off(FIELD *field, OPTIONS opts);
OPTIONS field_opts(FIELD *field);

DESCRIPTION set_field_opts() turns on the named options of field and turns off all remaining options. Options are boolean values that can be OR-ed together.

field_opts_on() turns on the named options; no other options are changed.

field_opts_off() turns off the named options; no other options are changed.

field_opts() returns the options set for field.

O_VISIBLE The field is displayed.
O_ACTIVE The field is visited during processing.
O_PUBLIC The field contents are displayed as data is entered.
O_EDIT The field can be edited.
O_WRAP Words not fitting on a line are wrapped to the next line.
O_BLANK The whole field is cleared if a character is entered in the first position.
O_AUTOSKIP Skip to the next field when the current field becomes full.
O_NULLOK A blank field is considered valid.
O_STATIC The field buffers are fixed in size.
O_PASSOK Validate field only if modified by user.

RETURN VALUES set_field_opts, field_opts_on and field_opts_off return one of the following:
E_OK The function returned successfully.
E_SYSTEM_ERROR System error.
E_CURRENT The field is the current field.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:
form_field_opts(3CURSES)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3CURSES), forms(3CURSES), attributes(5)

NOTES The header <form.h> automatically includes the headers <eti.h> and <curses.h>. 
FIELDTYPE *new_fieldtype(int (*field_check)(FIELD *, char *), int (*char_check)(int, char *));

int free_fieldtype(FIELDTYPE *fieldtype);

int set_fieldtype_arg(FIELDTYPE *fieldtype, char *(*mak_arg)(va_list *), char *(*copy_arg)(char *), void (*free_arg)(char *));

int set_fieldtype_choice(FIELDTYPE *fieldtype, int (*next_choice)(FIELD *, char *), int (*prev_choice)(FIELD *, char *));

FIELDTYPE *link_fieldtype(FIELDTYPE *type1, FIELDTYPE *type2);

new_fieldtype() creates a new field type. The application programmer must write the function field_check, which validates the field value, and the function char_check, which validates each character. free_fieldtype() frees the space allocated for the field type.

By associating function pointers with a field type, set_fieldtype_arg() connects to the field type additional arguments necessary for a set_field_type() call. Function mak_arg allocates a structure for the field specific parameters to set_field_type() and returns a pointer to the saved data. Function copy_arg duplicates the structure created by make_arg. Function free_arg frees any storage allocated by make_arg or copy_arg.

The form_driver() requests REQ_NEXT_CHOICE and REQ_PREV_CHOICE let the user request the next or previous value of a field type comprising an ordered set of values. set_fieldtype_choice() allows the application programmer to implement these requests for the given field type. It associates with the given field type those application-defined functions that return pointers to the next or previous choice for the field.

link_fieldtype() returns a pointer to the field type built from the two given types. The constituent types may be any application-defined or pre-defined types.

Routines that return pointers always return NULL on error. Routines that return an integer return one of the following:

- E_OK The function returned successfully.
- E_SYSTEM_ERROR System error.
- E_BAD_ARGUMENT An argument is incorrect.
- E_CONNECTED Type is connected to one or more fields.
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3CURSES), forms(3CURSES), attributes(5)

NOTES The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
form_field_userptr(3CURSES)

NAME
form_field_userptr, set_field_userptr, field_userptr – associate application data with forms

SYNOPSIS
c [ flag ... ] file ... -lform -lcurses [ library .. ]
#include <form.h>

int set_field_userptr(FIELD *field, char *ptr);
char *field_userptr(FIELD *field);

DESCRIPTION
Every field has an associated user pointer that can be used to store pertinent data.
set_field_userptr() sets the user pointer of field. field_userptr() returns the user pointer of field.

RETURN VALUES
field_userptr() returns NULL on error. set_field_userptr() returns one of the following:
E_OK The function returned successfully.
E_SYSTEM_ERROR System error.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), forms(3CURSES), attributes(5)

NOTES
The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
form_field_validation(3CURSES)

NAME
form_field_validation, set_field_type, field_type, field_arg – forms field data type validation

SYNOPSIS
c [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>

int set_field_type(FIELD *field, FIELDTYPE *type, ...);
FIELDTYPE *field_type(FIELD *field);
char *field_arg(FIELD *field);

DESCRIPTION
set_field_type() associates the specified field type with field. Certain field types take additional arguments. TYPE_ALNUM, for instance, requires one, the minimum width specification for the field. The other predefined field types are: TYPE_ALPHA, TYPE_ENUM, TYPE_INTEGER, TYPE_NUMERIC, and TYPE_REGEXP.

field_type() returns a pointer to the field type of field. NULL is returned if no field type is assigned.

field_arg() returns a pointer to the field arguments associated with the field type of field. NULL is returned if no field type is assigned.

RETURN VALUES
field_type() and field_arg() return NULL on error.

set_field_type() returns one of the following:
E_OK The function returned successfully.
E_SYSTEM_ERROR System error.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), forms(3CURSES), attributes(5)

NOTES
The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME
form_hook, set_form_init, form_init, set_form_term, form_term, set_field_init, field_init, set_field_term, field_term – assign application-specific routines for invocation by forms

SYNOPSIS
cc [-flag ...] file ... -lform -lcurses [ library ... ]
#include <form.h>

int set_form_init(FORM *form, void (*func)(FORM*));
void (*form_init)(FORM *form);

int set_form_term(FORM *form, void (*func)(FORM*));
void (*form_term)(FORM *form);

int set_field_init(FORM *form, void (*func)(FORM*));
void (*field_init)(FORM *form);

int set_field_term(FORM *form, void (*func)(FORM*));
void (*field_term)(FORM *form);

DESCRIPTION
These routines allow the programmer to assign application specific routines to be executed automatically at initialization and termination points in the forms application. The user need not specify any application-defined initialization or termination routines at all, but they may be helpful for displaying messages or page numbers and other chores.

set_form_init() assigns an application-defined initialization function to be called when the form is posted and just after a page change. form_init() returns a pointer to the initialization function, if any.

set_form_term() assigns an application-defined function to be called when the form is unposted and just before a page change. form_term() returns a pointer to the function, if any.

set_field_init() assigns an application-defined function to be called when the form is posted and just after the current field changes. field_init() returns a pointer to the function, if any.

set_field_term() assigns an application-defined function to be called when the form is unposted and just before the current field changes. field_term() returns a pointer to the function, if any.

RETURN VALUES
Routines that return pointers always return NULL on error. Routines that return an integer return one of the following:
E_OK The function returned successfully.
E_SYSTEM_ERROR System error.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

curses(3CURSES), forms(3CURSES), attributes(5)

NOTES

The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.
form_new(3CURSES)

NAME  form_new, new_form, free_form – create and destroy forms

SYNOPSIS  cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>
FORM *new_form(FIELD **fields);
int free_form(FORM *form);

DESCRIPTION  new_form() creates a new form connected to the designated fields and returns a
pointer to the form.

free_form() disconnects the form from its associated field pointer array and
deallocates the space for the form.

RETURN VALUES  new_form() always returns NULL on error. free_form() returns one of the
following:
E_OK The function returned successfully.
E_BAD_ARGUMENT An argument is incorrect.
E_POSTED The form is posted.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3CURSES), forms(3CURSES), attributes(5)

NOTES  The header <form.h> automatically includes the headers <eti.h> and
<curses.h>.
form_new_page(3CURSES)

NAME  form_new_page, set_new_page, new_page – forms pagination

SYNOPSIS  
```
cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>

int set_new_page(FIELD *field, int bool);
int new_page(FIELD *field);
```

DESCRIPTION  
set_new_page() marks field as the beginning of a new page on the form.

new_page() returns a boolean value indicating whether or not field begins a new page of the form.

RETURN VALUES  
new_page returns TRUE or FALSE.

set_new_page() returns one of the following:

- **E_OK** The function returned successfully.
- **E_CONNECTED** The field is already connected to a form.
- **E_SYSTEM_ERROR** System error.

ATTRIBUTES  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  
curses(3CURSES), forms(3CURSES), attributes(5)

NOTES  
The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
form_opts(3CURSES)

NAME    form_opts, set_form_opts, form_opts_on, form_opts_off – forms option routines

SYNOPSIS cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>

int set_form_opts(FORM *form, OPTIONS opts);
int form_opts_on(FORM *form, OPTIONS opts);
int form_opts_off(FORM *form, OPTIONS opts);

OPTIONS

form_opts(FORM *form);

DESCRIPTION set_form_opts() turns on the named options for form and turns off all remaining options. Options are boolean values which can be OR-ed together. form_opts_on() turns on the named options; no other options are changed. form_opts_off() turns off the named options; no other options are changed.

form_opts() returns the options set for form.

O_NL_OVERLOAD Overload the REQ_NEW_LINE form driver request.
O_BS_OVERLOAD Overload the REQ_DEL_PREV form driver request.

RETURN VALUES set_form_opts(), form_opts_on(), and form_opts_off() return one of the following:

E_OK The function returned successfully.
E_SYSTEM_ERROR System error.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3CURSES), forms(3CURSES), attributes(5)

NOTES The header <form.h> automatically includes the headers <eti.h> and <curses.h>.

210 man pages section 3: Curses Library Functions • Last Revised 31 Dec 1996
NAME
form_page, set_form_page, set_current_field, current_field, field_index – set forms current page and field

SYNOPSIS
```c
cc [ flag ... ] file ... -lform -lcurses [ library .. ]
#include <form.h>

int set_form_page(FORM *form, int page);
int form_page(FORM *form);
int set_current_field(FORM *form, FIELD *field);
FIELD *current_field(FORM *form);
int field_index(FIELD *field);
```

DESCRIPTION
`set_form_page()` sets the page number of `form` to `page`. `form_page()` returns the current page number of `form`.

`set_current_field()` sets the current field of `form` to `field`. `current_field()` returns a pointer to the current field of `form`.

`field_index()` returns the index in the field pointer array of `field`.

RETURN VALUES
`form_page()` returns -1 on error.

`current_field()` returns NULL on error.

`field_index()` returns -1 on error.

`set_form_page()` and `set_current_field()` return one of the following:

- **E_OK** The function returned successfully.
- **E_SYSTEM_ERROR** System error.
- **E_BAD_ARGUMENT** An argument is incorrect.
- **E_BAD_STATE** The routine was called from an initialization or termination function.
- **E_INVALID_FIELD** The field contents are invalid.
- **E_REQUEST_DENIED** The form driver request failed

ATTRIBUTES
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), forms(3CURSES), attributes(5)
The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>`. 
NAME
form_post, post_form, unpost_form – write or erase forms from associated subwindows

SYNOPSIS
cc [ flag ... ] file ... -lform -lcurses [ library .. ]
#include <form.h>

int post_form(FORM *form);

int unpost_form(FORM *form);

DESCRIPTION
post_form() writes form into its associated subwindow. The application programmer must use curses library routines to display the form on the physical screen or call update_panels() if the panels library is being used.

unpost_form() erases form from its associated subwindow.

RETURN VALUES
These routines return one of the following:

E_OK The function returned successfully.
E_SYSTEM_ERROR System error.
E_BAD_ARGUMENT An argument is incorrect.
E_POSTED The form is posted.
E_NOT_POSTED The form is not posted.
E_NO_ROOM The form does not fit in the subwindow.
E_BAD_STATE The routine was called from an initialization or termination function.
E_NOT_CONNECTED The field is not connected to a form.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), forms(3CURSES), panel_update(3CURSES), panels(3CURSES), attributes(5)

NOTES
The header <form.h> automatically includes the headers <eti.h> and <curses.h>.

Curses Library Functions 213
forms – character based forms package

#include <form.h>

The form library is built using the curses library, and any program using forms routines must call one of the curses initialization routines such as initscr. A program using these routines must be compiled with -lform and -lcurses on the cc command line.

The forms package gives the applications programmer a terminal-independent method of creating and customizing forms for user-interaction. The forms package includes: field routines, which are used to create and customize fields, link fields and assign field types; fieldtype routines, which are used to create new field types for validating fields; and form routines, which are used to create and customize forms, assign pre/post processing functions, and display and interact with forms.

The forms package establishes initial current default values for field attributes. During field initialization, each field attribute is assigned the current default value for that attribute. An application can change or retrieve a current default attribute value by calling the appropriate set or retrieve routine with a NULL field pointer. If an application changes a current default field attribute value, subsequent fields created using new_field() will have the new default attribute value. (The attributes of previously created fields are not changed if a current default attribute value is changed.)

The following table lists each forms routine and the name of the manual page on which it is described.

<table>
<thead>
<tr>
<th>Routine Name Index</th>
<th>Manual Page Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_field</td>
<td>form_page(3X)</td>
</tr>
<tr>
<td>data_ahead</td>
<td>form_data(3X)</td>
</tr>
<tr>
<td>data_behind</td>
<td>form_data(3X)</td>
</tr>
<tr>
<td>dup_field</td>
<td>form_field_new(3X)</td>
</tr>
<tr>
<td>dynamic_field_info</td>
<td>form_field_info(3X)</td>
</tr>
<tr>
<td>field_arg</td>
<td>form_field_validation(3X)</td>
</tr>
<tr>
<td>field_back</td>
<td>form_field_attributes(3X)</td>
</tr>
<tr>
<td>field_buffer</td>
<td>form_field_buffer(3X)</td>
</tr>
<tr>
<td>field_count</td>
<td>form_field(3X)</td>
</tr>
<tr>
<td>field_fore</td>
<td>form_field_attributes(3X)</td>
</tr>
<tr>
<td>field_index</td>
<td>form_page(3X)</td>
</tr>
<tr>
<td>field_info</td>
<td>form_field_info(3X)</td>
</tr>
<tr>
<td>field_init</td>
<td>form_hook(3X)</td>
</tr>
</tbody>
</table>

Last Revised 31 Dec 1996
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>field_just</td>
<td>form_field_just(3X)</td>
</tr>
<tr>
<td>field_opts</td>
<td>form_field_opts(3X)</td>
</tr>
<tr>
<td>field_opts_off</td>
<td>form_field_opts(3X)</td>
</tr>
<tr>
<td>field_opts_on</td>
<td>form_field_opts(3X)</td>
</tr>
<tr>
<td>field_pad</td>
<td>form_field_attributes(3X)</td>
</tr>
<tr>
<td>field_status</td>
<td>form_field_buffer(3X)</td>
</tr>
<tr>
<td>field_term</td>
<td>form_hook(3X)</td>
</tr>
<tr>
<td>field_type</td>
<td>form_field_validation(3X)</td>
</tr>
<tr>
<td>field_userptr</td>
<td>form_field_userptr(3X)</td>
</tr>
<tr>
<td>form_driver</td>
<td>form_driver(3X)</td>
</tr>
<tr>
<td>form_fields</td>
<td>form_field(3X)</td>
</tr>
<tr>
<td>form_init</td>
<td>form_hook(3X)</td>
</tr>
<tr>
<td>form_opts</td>
<td>form_opts(3X)</td>
</tr>
<tr>
<td>form_opts_off</td>
<td>form_opts(3X)</td>
</tr>
<tr>
<td>form_opts_on</td>
<td>form_opts(3X)</td>
</tr>
<tr>
<td>form_page</td>
<td>form_page(3X)</td>
</tr>
<tr>
<td>form_sub</td>
<td>form_win(3X)</td>
</tr>
<tr>
<td>form_term</td>
<td>form_hook(3X)</td>
</tr>
<tr>
<td>form_userptr</td>
<td>form_userptr(3X)</td>
</tr>
<tr>
<td>form_win</td>
<td>form_win(3X)</td>
</tr>
<tr>
<td>free_field</td>
<td>form_field_new(3X)</td>
</tr>
<tr>
<td>free_fieldtype</td>
<td>form_fieldtype(3X)</td>
</tr>
<tr>
<td>free_form</td>
<td>form_new(3X)</td>
</tr>
<tr>
<td>link_field</td>
<td>form_field_new(3X)</td>
</tr>
<tr>
<td>link_fieldtype</td>
<td>form_fieldtype(3X)</td>
</tr>
<tr>
<td>move_field</td>
<td>form_field(3X)</td>
</tr>
<tr>
<td>new_field</td>
<td>form_field_new(3X)</td>
</tr>
<tr>
<td>new_fieldtype</td>
<td>form_fieldtype(3X)</td>
</tr>
<tr>
<td>new_form</td>
<td>form_new(3X)</td>
</tr>
<tr>
<td>new_page</td>
<td>form_new_page(3X)</td>
</tr>
<tr>
<td>pos_form_cursor</td>
<td>form_cursor(3X)</td>
</tr>
</tbody>
</table>
forms(3CURSES)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>post_form</td>
<td>form_post(3X)</td>
</tr>
<tr>
<td>scale_form</td>
<td>form_win(3X)</td>
</tr>
<tr>
<td>set_current_field</td>
<td>form_page(3X)</td>
</tr>
<tr>
<td>set_field_back</td>
<td>form_field_attributes(3X)</td>
</tr>
<tr>
<td>set_field_buffer</td>
<td>form_field_buffer(3X)</td>
</tr>
<tr>
<td>set_field_fore</td>
<td>form_field_attributes(3X)</td>
</tr>
<tr>
<td>set_field_init</td>
<td>form_hook(3X)</td>
</tr>
<tr>
<td>set_field_just</td>
<td>form_field_just(3X)</td>
</tr>
<tr>
<td>set_field_opts</td>
<td>form_field_opts(3X)</td>
</tr>
<tr>
<td>set_field_pad</td>
<td>form_field_attributes(3X)</td>
</tr>
<tr>
<td>set_field_status</td>
<td>form_field_buffer(3X)</td>
</tr>
<tr>
<td>set_field_term</td>
<td>form_hook(3X)</td>
</tr>
<tr>
<td>set_field_type</td>
<td>form_field_validation(3X)</td>
</tr>
<tr>
<td>set_field_userptr</td>
<td>form_field_userptr(3X)</td>
</tr>
<tr>
<td>set_fieldtype_arg</td>
<td>form_fieldtype(3X)</td>
</tr>
<tr>
<td>set_fieldtype_choice</td>
<td>form_fieldtype(3X)</td>
</tr>
<tr>
<td>set_form_fields</td>
<td>form_field(3X)</td>
</tr>
<tr>
<td>set_form_init</td>
<td>form_hook(3X)</td>
</tr>
<tr>
<td>set_form_opts</td>
<td>form_opts(3X)</td>
</tr>
<tr>
<td>set_form_page</td>
<td>form_page(3X)</td>
</tr>
<tr>
<td>set_form_sub</td>
<td>form_win(3X)</td>
</tr>
<tr>
<td>set_form_term</td>
<td>form_hook(3X)</td>
</tr>
<tr>
<td>set_form_userptr</td>
<td>form_userptr(3X)</td>
</tr>
<tr>
<td>set_form_win</td>
<td>form_win(3X)</td>
</tr>
<tr>
<td>set_max_field</td>
<td>form_field_buffer(3X)</td>
</tr>
<tr>
<td>set_new_page</td>
<td>form_new_page(3X)</td>
</tr>
<tr>
<td>unpost_form</td>
<td>form_post(3X)</td>
</tr>
</tbody>
</table>

RETURN VALUES

Routines that return a pointer always return NULL on error. Routines that return an integer return one of the following:

- **E_OK** The function returned successfully.
- **E_CONNECTED** The field is already connected to a form.
forms(3CURSES)

E_SYSTEM_ERROR System error.
E_BAD_ARGUMENT An argument is incorrect.
E_CURRENT The field is the current field.
E_POSTED The form is posted.
E_NOT_POSTED The form is not posted.
E_INVALID_FIELD The field contents are invalid.
E_NOT_CONNECTED The field is not connected to a form.
E_NO_ROOM The form does not fit in the subwindow.
E_BAD_STATE The routine was called from an initialization or termination function.
E_REQUEST_DENIED The form driver request failed.
E_UNKNOWN_COMMAND An unknown request was passed to the form driver.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3CURSES), attributes(5) and 3X pages whose names begin "form_" for detailed routine descriptions.

NOTES The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
form_userptr(3CURSES)

NAME    form_userptr, set_form_userptr – associate application data with forms

SYNOPSIS cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>

int set_form_userptr(FORM *form, char *ptr);
char *form_userptr(FORM *form);

DESCRIPTION Every form has an associated user pointer that can be used to store pertinent data. set_form_userptr() sets the user pointer of form. form_userptr() returns the user pointer of form.

RETURN VALUES form_userptr() returns NULL on error. set_form_userptr() returns one of the following:
E_OK The function returned successfully.
E_SYSTEM_ERROR System error.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3CURSES), forms(3CURSES), attributes(5)

NOTES The header <form.h> automatically includes the headers <eti.h> and <curses.h>.
NAME

form_win, set_form_win, set_form_sub, form_sub, scale_form – forms window and
subwindow association routines

SYNOPSIS

cc [ flag ... ] file ... -lform -lcurses [ library ... ]
#include <form.h>

int set_form_win(FORM *form, WINDOW *win);
WINDOW *form_win(FORM *form);
int set_form_sub(FORM *form, WINDOW *sub);
WINDOW *form_sub(FORM *form);
int scale_form(FORM *form, int *rows, int *cols);

DESCRIPTION

set_form_win() sets the window of form to win. form_win() returns a pointer to
the window associated with form. set_form_sub() sets the subwindow of form to
sub. form_sub() returns a pointer to the subwindow associated with
form. scale_form() returns the smallest window size necessary for the subwindow
of form. rows and cols are pointers to the locations used to return the number of rows
and columns for the form.

RETURN VALUES

Routines that return pointers always return NULL on error. Routines that return an
integer return one of the following:

E_OK The function returned successfully.
E_SYSTEM_ERROR System error.
E_BAD_ARGUMENT An argument is incorrect.
E_NOT_CONNECTED The field is not connected to a form.
E_POSTED The form is posted.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

curses(3CURSES), forms(3CURSES), attributes(5)

NOTES

The header <form.h> automatically includes the headers <eti.h> and
<curses.h>.
The `getyx()` macro stores the current cursor position of the specified window in \( x \) and \( y \).

The `getparyx()` macro stores the \( x \) and \( y \) coordinates (relative to the parent window) of the specified window’s origin (upper-left corner). If `win` does not point to a subwindow, \( x \) and \( y \) are set to \(-1\).

The `getbegyx()` macro stores the \( x \) and \( y \) coordinates of the specified window’s origin (upper-left corner).

The `getmaxyx()` macro stores the numbers of rows in the specified window in \( y \) and the number of columns in \( x \).

**PARAMETERS**

- `win` Is a pointer to a window.
- `y` stores the \( y \) coordinate for the cursor or origin. The `getmaxyx()` macro uses it to store the number of rows in the window.
- `x` stores the \( x \) coordinate for the cursor or origin. The `getmaxyx()` macro uses it to store the number of columns in the window.

**RETURN VALUES**

These macros do not return a value.

**ERRORS**

None.
getcchar – get a wide character string (with rendition) from a cchar_t

#include <curses.h>

int getcchar(const cchar_t *wcval, wchar_t *wch, attr_t *attrs, short *color_pair, void *opt);

**DESCRIPTION**
If `wch` is not a null pointer, the `getcchar()` function splits the `cchar_t` object pointed to by `wcval` into a wide character string, attributes, and a color pair. It stores the attributes in the location pointed to by `attrs`, the color pair in the location pointed to by `color_pair`, and the wide character string in the location pointed to by `wch`.

If `wch` is a null pointer, the `getcchar()` function simply returns the number of wide characters in the `cchar_t` object pointed to by `wcval`. The objects pointed to by `attrs` and `color_pair` are not changed.

**PARAMETERS**
- `wcval` is a pointer to a `cchar_t` object.
- `wch` is a pointer to an object where a wide character string can be stored.
- `attrs` is a pointer to an object where attributes can be stored.
- `color_pair` is a pointer to an object where a color pair can be stored.
- `opt` is reserved for future use. Currently, this must be a null pointer.

**RETURN VALUES**
When `wch` is a null pointer, the `getcchar()` function returns the number of wide characters in the string pointed to by `wcval` including the null terminator.

When `wch` is not a null pointer, the `getcchar()` function returns `OK` on success and `ERR` otherwise.

**ERRORS**
None

**SEE ALSO**
`attroff(3XCURSES)`, `can_change_color(3XCURSES)`, `setcchar(3XCURSES)`
getch(3XCURSES)

NAME
getch, wgetch, mvgetch, mvwgetch — get a single-byte character from the terminal

SYNOPSIS
#include <curses.h>

int   getch(void);
int   wgetch(WINDOW *win);
int   mvgetch(int y, int x);
int   mvwgetch(WINDOW *win, int y, int x);

PARAMETERS

win  Is a pointer to the window associated with the terminal from which the
character is to be read.

y    Is the y (row) coordinate for the position of the character to be read.

x    Is the x (column) coordinate for the position of the character to be read.

DESCRIPTION

These functions read a single-byte character from the terminal associated with the
current or specified window. The results are unspecified if the input is not a
single-byte character. If keypad(3XCURSES) is enabled, these functions respond to the
pressing of a function key by returning the corresponding KEY_value defined in
<curses.h>

Processing of terminal input is subject to the general rules described on the
keypad(3XCURSES) manual page.

If echoing is enabled, then the character is echoed as though it were provided as an
input argument to addch(3XCURSES), except for the following characters:

<backspace> The input is interpreted as follows: unless the cursor already was
in column 0, <backspace> moves the cursor one column toward
the start of the current line and any characters after the
<backspace> are added or inserted starting there. The character at
the resulting cursor position it then deleted as though
delch(3XCURSES) were called, except that if the cursor was
originally in the first column of the line, the user is alerted as
though beep(3XCURSES) were called.

Function keys The user is alerted as though beep() were called. Information
concerning the function keys is not returned to the caller.

If the current or specified window is not a pad, and it has been moved modified since
the last refresh operation, then it will be refreshed before another character is read.

The following is a list of tokens for function keys that are returned by the getch() set
of functions if keypad handling is enabled (some terminals may not support all
tokens).

Constant Values

for Function Keys

222 man pages section 3: Curses Library Functions • Last Revised 4 May 1999
### Constant Description

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_BREAK</td>
<td>Break key</td>
</tr>
<tr>
<td>KEY_DOWN</td>
<td>The down arrow key</td>
</tr>
<tr>
<td>KEY_UP</td>
<td>The up arrow key</td>
</tr>
<tr>
<td>KEY_LEFT</td>
<td>The left arrow key</td>
</tr>
<tr>
<td>KEY_RIGHT</td>
<td>The right arrow key</td>
</tr>
<tr>
<td>KEY_HOME</td>
<td>Home key</td>
</tr>
<tr>
<td>KEY_BACKSPACE</td>
<td>Backspace</td>
</tr>
<tr>
<td>KEY_F0</td>
<td>Function keys. Space for 64 keys is reserved.</td>
</tr>
<tr>
<td>KEY_F(n)</td>
<td>For 0 &lt;= n &lt;= 63</td>
</tr>
<tr>
<td>KEY_DL</td>
<td>Delete line</td>
</tr>
<tr>
<td>KEY_IL</td>
<td>Insert line</td>
</tr>
<tr>
<td>KEY_DC</td>
<td>Delete character</td>
</tr>
<tr>
<td>KEY_IC</td>
<td>Insert char or enter insert mode</td>
</tr>
<tr>
<td>KEY_EIC</td>
<td>Exit insert char mode</td>
</tr>
<tr>
<td>KEY_CLEAR</td>
<td>Clear screen</td>
</tr>
<tr>
<td>KEY_EOS</td>
<td>Clear to end of screen</td>
</tr>
<tr>
<td>KEY_EOL</td>
<td>Clear to end of line</td>
</tr>
<tr>
<td>KEY_SF</td>
<td>Scroll 1 line forward</td>
</tr>
<tr>
<td>KEY_SR</td>
<td>Scroll 1 line backwards</td>
</tr>
<tr>
<td>KEY_NPAGE</td>
<td>Next page</td>
</tr>
<tr>
<td>KEY_PPAGE</td>
<td>Previous page</td>
</tr>
<tr>
<td>KEY_STAB</td>
<td>Set tab</td>
</tr>
<tr>
<td>KEY_CTAB</td>
<td>Clear tab</td>
</tr>
<tr>
<td>KEY_CATAB</td>
<td>Clear all tabs</td>
</tr>
<tr>
<td>KEY_ENTER</td>
<td>Enter or send</td>
</tr>
<tr>
<td>KEY_SRESET</td>
<td>Soft (partial) reset</td>
</tr>
<tr>
<td>KEY_RESET</td>
<td>Reset or hard reset</td>
</tr>
<tr>
<td>KEY_PRINT</td>
<td>Print or copy</td>
</tr>
<tr>
<td>KEY_LL</td>
<td>Home down or bottom (lower left)</td>
</tr>
<tr>
<td>Constant</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>KEY_A1</td>
<td>Upper left of keypad</td>
</tr>
<tr>
<td>KEY_A3</td>
<td>Upper right of keypad</td>
</tr>
<tr>
<td>KEY_B2</td>
<td>Center of keypad</td>
</tr>
<tr>
<td>KEY_C1</td>
<td>Lower left of keypad</td>
</tr>
<tr>
<td>KEY_C3</td>
<td>Lower right of keypad</td>
</tr>
<tr>
<td>KEY_BTAB</td>
<td>Back tab</td>
</tr>
<tr>
<td>KEY_BEG</td>
<td>Beginning key</td>
</tr>
<tr>
<td>KEY_CANCEL</td>
<td>Cancel key</td>
</tr>
<tr>
<td>KEY_CLOSE</td>
<td>Close key</td>
</tr>
<tr>
<td>KEY_COMMAND</td>
<td>Cmd (command) key</td>
</tr>
<tr>
<td>KEY_COPY</td>
<td>Copy key</td>
</tr>
<tr>
<td>KEY_CREATE</td>
<td>Create key</td>
</tr>
<tr>
<td>KEY_END</td>
<td>End key</td>
</tr>
<tr>
<td>KEY_EXIT</td>
<td>Exit key</td>
</tr>
<tr>
<td>KEY_FIND</td>
<td>Find key</td>
</tr>
<tr>
<td>KEY_HELP</td>
<td>Help key</td>
</tr>
<tr>
<td>KEY_MARK</td>
<td>Mark key</td>
</tr>
<tr>
<td>KEY_MESSAGE</td>
<td>Message key</td>
</tr>
<tr>
<td>KEY_MOVE</td>
<td>Move key</td>
</tr>
<tr>
<td>KEY_NEXT</td>
<td>Next object key</td>
</tr>
<tr>
<td>KEY_OPEN</td>
<td>Open key</td>
</tr>
<tr>
<td>KEY_OPTIONS</td>
<td>Options key</td>
</tr>
<tr>
<td>KEY_PREVIOUS</td>
<td>Previous object key</td>
</tr>
<tr>
<td>KEY_REDO</td>
<td>Redo key</td>
</tr>
<tr>
<td>KEY_REFERENCE</td>
<td>Reference key</td>
</tr>
<tr>
<td>KEY_REFRESH</td>
<td>Refresh key</td>
</tr>
<tr>
<td>KEY_REPLACE</td>
<td>Replace key</td>
</tr>
<tr>
<td>KEY_RESTART</td>
<td>Restart key</td>
</tr>
<tr>
<td>KEY_RESUME</td>
<td>Resume key</td>
</tr>
<tr>
<td>Constant</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>KEY_SAVE</td>
<td>Save key</td>
</tr>
<tr>
<td>KEY_SBEG</td>
<td>Shifted beginning key</td>
</tr>
<tr>
<td>KEY_SCANCEL</td>
<td>Shifted cancel key</td>
</tr>
<tr>
<td>KEY_SCOMMAND</td>
<td>Shifted command key</td>
</tr>
<tr>
<td>KEY_SCOPY</td>
<td>Shifted copy key</td>
</tr>
<tr>
<td>KEY_SCREATE</td>
<td>Shifted create key</td>
</tr>
<tr>
<td>KEY_SDC</td>
<td>Shifted delete char key</td>
</tr>
<tr>
<td>KEY SDL</td>
<td>Shifted delete line key</td>
</tr>
<tr>
<td>KEY_SELECT</td>
<td>Select key</td>
</tr>
<tr>
<td>KEY_SEND</td>
<td>Shifted end key</td>
</tr>
<tr>
<td>KEY_SEOL</td>
<td>Shifted clear line key</td>
</tr>
<tr>
<td>KEY_SEXIT</td>
<td>Shifted exit key</td>
</tr>
<tr>
<td>KEY_SFIN</td>
<td>Shifted find key</td>
</tr>
<tr>
<td>KEY_SHELP</td>
<td>Shifted help key</td>
</tr>
<tr>
<td>KEY_SHOME</td>
<td>Shifted home key</td>
</tr>
<tr>
<td>KEY_SIC</td>
<td>Shifted input key</td>
</tr>
<tr>
<td>KEY_SLEFT</td>
<td>Shifted left arrow key</td>
</tr>
<tr>
<td>KEY_SMESSAGES</td>
<td>Shifted messages key</td>
</tr>
<tr>
<td>KEY_SMOVE</td>
<td>Shifted move key</td>
</tr>
<tr>
<td>KEY_SNEXT</td>
<td>Shifted next key</td>
</tr>
<tr>
<td>KEY_SOPTIONS</td>
<td>Shifted options key</td>
</tr>
<tr>
<td>KEY_SPREVIOUS</td>
<td>Shifted previous key</td>
</tr>
<tr>
<td>KEY_SPRINT</td>
<td>Shifted print key</td>
</tr>
<tr>
<td>KEY_SREDO</td>
<td>Shifted redo key</td>
</tr>
<tr>
<td>KEY_SREPLACE</td>
<td>Shifted replace key</td>
</tr>
<tr>
<td>KEY_SRIGHT</td>
<td>Shifted right arrow key</td>
</tr>
<tr>
<td>KEY_SRESUME</td>
<td>Shifted resume key</td>
</tr>
<tr>
<td>KEY_SSSAVE</td>
<td>Shifted save key</td>
</tr>
<tr>
<td>KEY_SSUSPEND</td>
<td>Shifted suspend key</td>
</tr>
</tbody>
</table>

getch(3XCURSES)
getch(3XCURSES)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_SUNDO</td>
<td>Shifted undo key</td>
</tr>
<tr>
<td>KEY_SUSPEND</td>
<td>Suspend key</td>
</tr>
<tr>
<td>KEY_UNDO</td>
<td>Undo key</td>
</tr>
</tbody>
</table>

**RETURN VALUES**
Upon successful completion, these functions return the single-byte character, `KEY_`
value, or `ERR`. When in the nodelay mode and no data is available, `ERR` is returned.

**ERRORS**
No errors are defined.

**USAGE**
Applications should not define the escape key by itself as a single-character function.

When using these functions, nocbreak mode (`cbreak(3XCURSES)`) and echo mode (`echo(3XCURSES)`) should not be used at the same time. Depending on the state of the terminal when each character is typed, the application may produce undesirable results.

**SEE ALSO**
cbreak(3XCURSES), echo(3XCURSES), halfdelay(3XCURSES), keypad(3XCURSES), nodelay(3XCURSES), notimeout(3XCURSES), raw(3XCURSES), timeout(3XCURSES)
getnstr(3XCURSES)

NAME
getnstr, getstr, mvgetnstr, mvgetstr, mvwgetnstr, mvwgetstr, wgetnstr, wgetstr – get a multibyte character string from terminal

SYNOPSIS
#include <curses.h>

int getnstr(char *str, int n);
int getstr(char *str);
int mvgetnstr(int y, int x, char *str, int n);
int mvgetstr(int y, int x, char *str);
int mvwgetnstr(WINDOW *win, int y, int x, char *str, int n);
int mvwgetstr(WINDOW *win, int y, int x, char *str);
int wgetnstr(WINDOW *win, char *str, int n);
int wgetstr(WINDOW *win, char *str);

DESCRIPTION

The getstr() and wgetstr() functions get a character string from the terminal associated with the window stdscr or window win, respectively. The mvgetstr() and mvwgetstr() functions move the cursor to the position specified in stdscr or win, respectively, then get a character string.

These functions call wgetch(3XCURSES) and place each received character in str until a newline is received, which is also placed in str. The erase and kill characters set by the user are processed.

The getnstr(), mvgetnstr(), mvwgetnstr() and wgetnstr() functions read at most n characters. These functions are used to prevent overflowing the input buffer.

The getnstr(), wgetnstr(), mvgetnstr(), and mvwgetnstr() functions only return complete multibyte characters. If the area pointed to by str is not large enough to hold at least one character, these functions fail.

PARAMETERS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Is a pointer to the area where the character string is to be placed.</td>
</tr>
<tr>
<td>n</td>
<td>Is the maximum number of characters to read from input.</td>
</tr>
<tr>
<td>y</td>
<td>Is the y (row) coordinate of starting position of character string to be read.</td>
</tr>
<tr>
<td>x</td>
<td>Is the x (column) coordinate of starting position of character string to be read.</td>
</tr>
<tr>
<td>win</td>
<td>Points to the window associated with the terminal from which the character is to be read.</td>
</tr>
</tbody>
</table>

RETURN VALUES

On success, these functions return OK. Otherwise, they return ERR.

ERRORS

None.
SEE ALSO

getnstr(3XCURSES)
getn_wstr(3XCURSES)

NAME
getn_wstr, get_wstr, mvgetn_wstr, mvget_wstr, mvwgetn_wstr, mvwget_wstr, wgetn_wstr, wget_wstr

SYNOPSIS
#include <curses.h>

int getn_wstr(wint_t *wstr, int, int get_wstr(wint_t *wstr);
int mvgetn_wstr(int y, int x, wint_t *wstr, int n);
int mvget_wstr(int y, int x, wint_t *wstr);
int mvwgetn_wstr(WINDOW *win, int y, int x, wint_t *wstr, int n);
int mvwget_wstr(WINDOW *win, int y, int x, wint_t *wstr);
int wgetn_wstr(WINDOW *win, wint_t *wstr, int n);
int wget_wstr(WINDOW *win, wint_t *wstr);

DESCRIPTION
The get_wstr() and wget_wstr() functions get a wide character string from the
terminal associated with the window stdscr or window win, respectively. The
mvget_wstr() and mvwget_wstr() functions move the cursor to the position
specified in stdscr or win, respectively, then get a wide character string.

These functions call wget_wch(3XCURSES) and place each received character in wstr
until a newline character, end-of-line character, or end-of-file character is received,
which is also placed in wstr. The erase and kill characters set by the user are processed.

The getn_wstr(), mvgetn_wstr(), mvwgetn_wstr() and wgetn_wstr() functions read at most n characters. These functions are used to prevent overflowing
the input buffer.

PARAMETERS
wstr    Is a pointer to the area where the character string is to be placed.
n      Is the maximum number of characters to read from input.
y    Is the y (row) coordinate of starting position of character string to be read.
x    Is the x (column) coordinate of starting position of character string to be read.
win     points to the window associated with the terminal from which the
character is to be read.

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None.

SEE ALSO
getch(3XCURSES), getnstr(3XCURSES)
The `get_wch()` and `wget_wch()` functions get a wide character from the terminal associated with the window `stdscr` or window `win`, respectively. The `mvget_wch()` and `mvwget_wch()` functions move the cursor to the position specified in `stdscr` or `win`, respectively, then get a character.

If the window is not a pad and has been changed since the last call to `refresh()` (3XCURSES), `get_wch()` calls `refresh()` to update the window before the next character is read.

The setting of certain functions affects the behavior of the `get_wch()` set of functions. For example, if `cbreak()` (3XCURSES) is set, characters typed by the user are immediately processed. If `halfdelay()` (3XCURSES) is set, `get_wch()` waits until a character is typed or returns ERR if no character is typed within the specified timeout period. This timeout can also be specified for individual windows with the `delay` parameter of `timeout()` (3XCURSES) A negative value waits for input; a value of 0 returns ERR if no input is ready; a positive value blocks until input arrives or the time specified expires (in which case ERR is returned). If `nodelay()` (3XCURSES) is set, ERR is returned if no input is waiting; if not set, `get_wch()` waits until input arrives. Each character will be echoed to the window unless `noecho()` (3XCURSES) has been set.

If keypad handling is enabled (keypad() (3XCURSES) is TRUE), the token for the function key (a KEY_ value) is stored in the object pointed to by `ch` and KEY_CODE_YES is returned. If a character is received that could be the beginning of a function key (for example, ESC), an inter-byte timer is set. If the remainder of the sequence is not received before the time expires, the character is passed through; otherwise, the value of the function key is returned. If `notimeout()` is set, the inter-byte timer is not used.

The ESC key is typically a prefix key used with function keys and should not be used as a single character.

See the `getch()` (3XCURSES) manual page for a list of tokens for function keys that are returned by the `get_wch()` set of functions if keypad handling is enabled (Some terminals may not support all tokens).

**Parameters**

- `ch` is a pointer to a wide integer where the returned wide character or KEY_ value can be stored.
- `win` is a pointer to the window associated with the terminal from which the character is to be read.
get_wch(3XCURSES)

| y | Is the y (row) coordinate for the position of the character to be read. |
| x | Is the x (column) coordinate for the position of the character to be read. |

**RETURN VALUES**
When these functions successfully report the pressing of a function key, they return KEY_CODE_YES. When they successfully report a wide character, they return OK. Otherwise, they return ERR.

**ERRORS**
None.

**SEE ALSO**
cbreak(3XCURSES), echo(3XCURSES), halfdelay(3XCURSES), keypad(3XCURSES), nodelay(3XCURSES), notimeout(3XCURSES), raw(3XCURSES), timeout(3XCURSES)
getwin(3XCURSES)

NAME
getwin, putwin – read a window from, and write a window to, a file

SYNOPSIS
#include <curses.h>

WINDOW *getwin(FILE *filep);

int putwin(WINDOW *win, FILE *filep);

DESCRIPTION
The getwin() function reads window-related data (written earlier by putwin()) from the stdio stream pointed to by filep. It then creates and initializes a new window using that data.

The putwin() function writes all the data associated with the window pointed to by win to the stdio stream pointed to by filep. The getwin() function can later retrieve this data.

PARAMETERS
filep Is a pointer to a stdio stream.
win Is a pointer to a window.

RETURN VALUES
On success, the getwin() function returns a pointer to the new window created. Otherwise, it returns a null pointer.

On success, the putwin() function returns OK. Otherwise, it returns ERR.

ERRORS
None.

SEE ALSO
scr_dump(3XCURSES)
NAME  
halfdelay – enable/disable half-delay mode

SYNOPSIS
#include <curses.h>

int halfdelay(int tenths);

DESCRIPTION
The halfdelay() function is similar to cbreak(3XCURSES) in that when set, characters typed by the user are immediately processed by the program. The difference is that ERR is returned if no input is received after tenths tenths seconds.

The nocbreak(3XCURSES) function should be used to leave half-delay mode.

PARAMETERS
tenths  Is the number of tenths of seconds for which to block input (1 to 255).

RETURN VALUES
On success, the halfdelay() function returns OK. Otherwise, it returns ERR.

ERRORS
None.

SEE ALSO
cbreak(3XCURSES)
has_ic, has_il
– determine insert/delete character/line capability

#include <curses.h>

bool has_ic(void);
bool has_il(void);

DESCRIPTION
The has_ic() function determines whether or not the terminal has insert/delete character capability.
The has_il() function determines whether or not the terminal has insert/delete line capability.

RETURN VALUES
The has_ic() function returns TRUE if the terminal has insert/delete character capability and FALSE otherwise.
The has_il() function returns TRUE if the terminal has insert/delete line capability and FALSE otherwise.

ERRORS
None.
hline(3XCURSES)

NAME
hline, mvhline, mvvline, mvwhline, mvwvline, vline, whline, wvline – use single-byte characters (and renditions) to draw lines

SYNOPSIS
#include <curses.h>

int hline(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 vline(chtype ch, int n);
int whline(WINDOW *win, chtype ch, int n);
int wvline(WINDOW *win, chtype ch, int n);

DESCRIPTION
The hline(), vline(), whline(), wvline() functions draw a horizontal or vertical line, in either the window stdscr or win starting at the current cursor position. The line is drawn using the character ch and is a maximum of n positions long, or as many as will fit into the window. If ch is 0 (zero), the default horizontal or vertical character is used.

The mvhline(), mvvline(), mvwhline(), mvwvline() functions are similar to the previous group of functions but the line begins at cursor position specified by x and y.

The functions with names ending with hline() draw horizontal lines proceeding towards the last column of the same line. The functions with names ending with vline() draw vertical lines proceeding towards the last column of the same line.

These functions do not change the position of the cursor.

PARAMETERS
ch Is the character used to draw the line.
n Is the maximum number of characters in the line.
y Is the y (row) coordinate for the start of the line.
x Is the x (column) coordinate for the start of the line.
win Is a pointer to a window.

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None

SEE ALSO
border(3XCURSES), border_set(3XCURSES), hline_set(3XCURSES)
hline_set(3XCURSES)

NAME

hline_set, mvhline_set, mvvline_set, mvwhline_set, mvwvline_set, vline_set,
whline_set, wvline_set – use complex characters (and renditions) to draw lines

SYNOPSIS

#include <curses.h>

int hline_set(const cchar_t *ch, 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);
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);

DESCRIPTION

The hline_set(), vline_set(), whline_set(), wvline_set() functions draw a line, in either the window stdscr or win starting at the current cursor position. The line is drawn using the character wch and is a maximum of n positions long, or as many as will fit into the window. If wch is a null pointer, the default horizontal or vertical character is used.

The mvhline_set(), mvvline_set(), mvwhline_set(), mvwvline_set() functions are similar to the previous group of functions but the line begins at cursor position specified by x and y.

The functions with names ending with hline_set() draw horizontal lines proceeding towards the last column of the same line. The functions with names ending with vline_set() draw vertical lines proceeding towards the last column of the same line.

These functions do not change the position of the cursor.

PARAMETERS

wch Is the complex character used to draw the line.

n Is the maximum number of characters in the line.

y Is the y (row) coordinate for the start of the line.

x Is the x (column) coordinate for the start of the line.

win Is a pointer to a window.

RETURN VALUES

On success, these functions return OK. Otherwise, they return ERR.

ERRORS

None.

SEE ALSO

border(3XCURSES), border_set(3XCURSES), hline(3XCURSES)
idcok — enable/disable hardware insert-character and delete-character features

SYNOPSIS

```c
#include <curses.h>

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

DESCRIPTION

The `idcok()` function enables or disables the use of hardware insert-character and delete-character features in `win`. If `bf` is set to `TRUE`, the use of these features in `win` is enabled (if the terminal is equipped). If `bf` is set to `FALSE`, their use in `win` is disabled.

PARAMETERS

- `win` Is a pointer to a window.
- `bf` Is a Boolean expression.

RETURN VALUES

The `idcok()` function does not return a value.

ERRORS

None.

SEE ALSO

`clearok(3XCURSES)`, `doupdate(3XCURSES)`
immedok(3XCURSES)

<table>
<thead>
<tr>
<th>NAME</th>
<th>immedok – call refresh on changes to window</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>#include &lt;curses.h&gt;</td>
</tr>
<tr>
<td></td>
<td>int immedok(WINDOW *win, bool bf);</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>If bf is TRUE, immedok() calls refresh(3XCURSES) if any change to the window image is made (for example, through functions such as addch(3XCURSES), clrtoeol(3XCURSES), and scr(3XCURSES)). Repeated calls to refresh() may affect performance negatively. The immedok() function is disabled by default.</td>
</tr>
<tr>
<td>PARAMETERS</td>
<td>win Is a pointer to the window that is to be refreshed.</td>
</tr>
<tr>
<td></td>
<td>bf Is a Boolean expression.</td>
</tr>
<tr>
<td>RETURN VALUES</td>
<td>The immedok() function does not return a value.</td>
</tr>
<tr>
<td>ERRORS</td>
<td>None.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>addch(3XCURSES), clearok(3XCURSES), clrtoeol(3XCURSES), doupdate(3XCURSES), scr(3XCURSES)</td>
</tr>
</tbody>
</table>
### NAME
inch, mvinch, mvwinch, winch – return a single-byte character (with rendition)

### SYNOPSIS
```c
#include <curses.h>

chtype inch(void);
chtype mvinch(int y, int x);
chtype mvwinch(WINDOW *win, int y, int x);
chtype winch(WINDOW *win);
```

### DESCRIPTION
The `inch()` and `winch()` functions return the `chtype` character located at the current cursor position of the `stdscr` window and window `win`, respectively. The `mvinch()` and `mvwinch()` functions return the `chtype` character located at the position indicated by the `x` (column) and `y` (row) parameters (the former in the `stdscr` window; the latter in window `win`).

The complete character/attribute pair will be returned. The character or attributes can be extracted by performing a bitwise AND on the returned value, using the constants `A_CHARTEXT`, `A_ATTRIBUTES`, and `A_COLOR`.

### PARAMETERS
- `y` Is the `y` (row) coordinate of the position of the character to be returned.
- `x` Is the `x` (column) coordinate of the position of the character to be returned.
- `win` Is a pointer to the window that contains the character to be returned.

### RETURN VALUES
On success, these functions return the specified character and rendition. Otherwise, they return `ERR`.

### ERRORS
None.

### SEE ALSO
`addch(3XCURSES)`, `attroff(3XCURSES)`
The `inchstr()` and `winchstr()` functions retrieve the character string (with rendition) starting at the current cursor position of the `stdscr` window and window `win`, respectively, and ending at the right margin. The `mvinchstr()` and `mvwinchstr()` functions retrieve the character string located at the position indicated by the `x` (column) and `y` (row) parameters (the former in the `stdscr` window; the latter in window `win`).

The `inchnstr()`, `winchnstr()`, `mvinchnstr()`, and `mvwinchnstr()` functions retrieve at most `n` characters from the window `stdscr` and `win`, respectively. The former two functions retrieve the string, starting at the current cursor position; the latter two commands retrieve the string, starting at the position specified by the `x` and `y` parameters.

All these functions store the retrieved character string in the object pointed to by `chstr`.

The complete character/attribute pair is retrieved. The character or attributes can be extracted by performing a bitwise AND on the retrieved value, using the constants `A_CHARTEXT`, `A_ATTRIBUTES`, and `A_COLOR`. The character string can also be retrieved without attributes by using `instr(3XCURSES)` set of functions.

**PARAMETERS**

- `chstr` Is a pointer to an object that can hold the retrieved character string.
- `n` Is the number of characters not to exceed when retrieving `chstr`.
- `y` Is the `y` (row) coordinate of the starting position of the string to be retrieved.
- `x` Is the `x` (column) coordinate of the starting position of the string to be retrieved.
- `win` Is a pointer to the window in which the string is to be retrieved.

**RETURN VALUES**

On success, these functions return `OK`. Otherwise, they return `ERR`. 

---

240  man pages section 3: Curses Library Functions • Last Revised 1 Jun 1996
ERRORS | None.
SEE ALSO | inch(3XCURSES), innstr(3XCURSES)

inchnstr(3XCURSES)
initscr(3XCURSES)

NAME

initscr, newterm – screen initialization functions

SYNOPSIS

#include <curses.h>

WINDOW *initscr(void);
SCREEN *newterm(char *type, FILE *outfp, FILE *infp);

PARAMETERS

type Is a string defining the terminal type to be used in place of TERM.
outfp Is a pointer to a file to be used for output to the terminal.
infp Is the pointer to a file to be used for input to the terminal.

DESCRIPTION

The initscr() function initializes X/Open Curses data structures, determines the terminal type, and ensures the first call to refresh(3XCURSES) clears the screen.

The newterm() function opens a new terminal with each call. It should be used instead of initscr() when the program interacts with more than one terminal. It returns a variable of type SCREEN, which should be used for later reference to that terminal. Before program termination, endwin() should be called for each terminal.

The only functions that you can call before calling initscr() or newterm() are filter(3XCURSES), ripoffline(3XCURSES), slk_init(3XCURSES), and use_env(3XCURSES).

RETURN VALUES

On success, the initscr() function returns a pointer to stdscr; otherwise, initscr() does not return.

On success, the newterm() function returns a pointer to the specified terminal; otherwise, a null pointer is returned.

ERRORS

None.

SEE ALSO

del_curterm(3XCURSES), delscreen(3XCURSES), doupdate(3XCURSES), endwin(3XCURSES), filter(3XCURSES), slk_attroff(3XCURSES), use_env(3XCURSES)
### NAME
innstr, instr, mvinnstr, mvinstr, mvwinnstr, mvwinstr, winstr, winstr – retrieve a multibyte character string (without rendition)

### SYNOPSIS
```c
#include <curses.h>

int innstr(char *str, int n);
int instr(char *str);
int mvinnstr(int y, int x, char *str, int n);
int mvinstr(int y, int x, char *str);
int mvwinnstr(WINDOW *win, int y, int x, char *str, int n);
int mvwinstr(WINDOW *win, int y, int x, char *str);
int winstr(WINDOW *win, char *str);
int winnstr(WINDOW *win, char *str, int n);
```

### PARAMETERS
- **str**
  - Is a pointer to an object that can hold the retrieved multibyte character string.
- **n**
  - Is the number of characters not to exceed when retrieving `str`.
- **y**
  - Is the y (row) coordinate of the starting position of the string to be retrieved.
- **x**
  - Is the x (column) coordinate of the starting position of the string to be retrieved.
- **win**
  - Is a pointer to the window in which the string is to be retrieved.

### DESCRIPTION
The `inst` and `winstr` functions retrieve a multibyte character string (without attributes) starting at the current cursor position of the `stdscr` window and window `win`, respectively, and ending at the right margin. The `mvinnstr()` and `mvwinnstr()` functions retrieve a multibyte character string located at the position indicated by the `x` (column) and `y` (row) parameters (the former in the `stdscr` window; the latter in window `win`).

The `innstr()`, `winnstr()`, `mvinstr()`, and `mvwinnstr()` functions retrieve at most `n` characters from the window `stdscr` and `win`, respectively. The former two functions retrieve the string starting at the current cursor position; the latter two commands return the string, starting at the position specified by the `x` and `y` parameters.

All these functions store the retrieved string in the object pointed to by `str`. They only store complete multibyte characters. If the area pointed to by `str` is not large enough to hold at least one character, these functions fail.
innstr(3XCURSES)

Only the character portion of the character/rendition pair is returned. To return the complete character/rendition pair, use winchstr().

| ERRORS   | OK          | Successful completion. |
|          | ERR         | An error occurred.     |

| USAGE    | All functions except winnstr() may be macros. |

| SEE ALSO | inch(3XCURSES), inchstr(3XCURSES) |
innwstr(3XCURSES)

NAME
innwstr, inwstr, mvinnwstr, mvinwstr, mvwinnwstr, mvwinwstr, winwstr, winstr –
retrieve a wide character string (without rendition)

SYNOPSIS
#include <curses.h>

int innwstr(wchar_t *wstr, int n);
int inwstr(wchar_t *wstr);
int mvinnwstr(int y, int x, wchar_t *wstr, int n);
int mvinwstr(int y, int x, wchar_t *wstr);
int mvwinnwstr(WINDOW*win, int y, int x, wchar_t *wstr, int n);
int mvwinwstr(WINDOW*win, int y, int x, wchar_t *wstr);
int winwstr(WINDOW*win, wchar_t *wstr);
int winnstr(WINDOW*win, wchar_t *wstr, int n);

PARAMETERS
wstr Is a pointer to an object that can hold the retrieved
multibyte character string.

n Is the number of characters not to exceed when
retrieving wstr.

y Is the y (row) coordinate of the starting position of the
string to be retrieved.

x Is the x (column) coordinate of the starting position of
the string to be retrieved.

win Is a pointer to the window in which the string is to be
retrieved.

DESCRIPTION
The inwstr() and winwstr() functions retrieve a wide character string (without
attributes) starting at the current cursor position of the stdscr window and window
win, respectively, and ending at the right margin. The mvinwstr() and
mvwinwstr() functions retrieve a wide character string located at the position
indicated by the x (column) and y (row) parameters (the former in the stdscr
window; the latter in window win).

The innwstr(), winnstr(), mvinnwstr(), and mvwinnwstr() functions
retrieve at most n characters from the window stdscr and win, respectively. The
former two functions retrieve the string starting at the current cursor position; the
latter two commands return the string, starting at the position specified by the x and y
parameters.

All these functions store the retrieved string in the object pointed to by wstr. They only
store complete wide characters. If the area pointed to by wstr is not large enough to
hold at least one character, these functions fail.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>innwstr(3XCURSES)</td>
<td>Only the character portion of the character/rendition pair is returned. To return the complete character/rendition pair, use win_wchstr(3XCURSES).</td>
</tr>
<tr>
<td>RETURN VALUES</td>
<td>On success, the innwstr(), mvinnwstr(), mvwinwstr(), and winwstr() functions return OK. Otherwise, they return ERR.</td>
</tr>
<tr>
<td></td>
<td>On success, the innwstr(), mvinnwstr(), mvwinwstr(), and winwstr() functions return the number of characters read into the string. Otherwise, they return ERR.</td>
</tr>
<tr>
<td>ERRORS</td>
<td>None.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>in_wch(3XCURSES), in_wchnstr(3XCURSES)</td>
</tr>
</tbody>
</table>
NAME
insch, winsch, mvinsch, mvwinsch – insert a character

SYNOPSIS
#include <curses.h>

int insch(chtype ch);
int mvinsch(int y, int x, chtype ch);
int mvwinsch(WINDOW *win, int y, int x, chtype ch);
int winsch(WINDOW *win, chtype ch);

PARAMETERS
ch Is the character to be inserted.
y Is the y (row) coordinate of the position of the
   character.
x Is the x (column) coordinate of the position of the
   character.
win Is a pointer to the window in which the character is to
     be inserted.

DESCRIPTION
These functions insert the character and rendition from ch into the current or specified
window at the current or specified position.

These functions do not perform wrapping and do not advance the cursor position.
These functions perform special-character processing, with the exception that if a
newline is inserted into the last line of a window and scrolling is not enabled, the
behavior is unspecified.

RETURN VALUES
Upon successful completion, these functions return OK. Otherwise, they return ERR.

ERRORS
No errors are defined.

USAGE
These functions are only guaranteed to operate reliably on character sets in which each
character fits into a single byte, whose attributes can be expressed using only constants
with the A_ prefix.

SEE ALSO
ins_wch(3XCURSES)
### insdelln(3XCURSES)

<table>
<thead>
<tr>
<th>NAME</th>
<th>insdelln, winsdelln – insert/delete lines to/from the window</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>#include &lt;curses.h&gt;</td>
</tr>
<tr>
<td></td>
<td>int insdelln(int n);</td>
</tr>
<tr>
<td></td>
<td>int winsdelln(WINDOW *win, int n);</td>
</tr>
<tr>
<td>PARAMETERS</td>
<td>n Is the number of lines to insert or delete (positive n inserts; negative n deletes).</td>
</tr>
<tr>
<td></td>
<td>win Is a pointer to the window in which to insert or delete a line.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The insdelln() and winsdelln() functions insert or delete blank lines in stdscr or win, respectively. When n is positive, n lines are added before the current line and the bottom n lines are lost; when n is negative, n lines are deleted starting with the current line, the remaining lines are moved up, and the bottom n lines are cleared. The position of the cursor does not change.</td>
</tr>
<tr>
<td>RETURN VALUES</td>
<td>On success, these functions return OK. Otherwise, they return ERR.</td>
</tr>
<tr>
<td>ERRORS</td>
<td>None.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>deleteln(3XCURSES), insertln(3XCURSES)</td>
</tr>
</tbody>
</table>
NAME  insertln, winsertln – insert a line in a window

SYNOPSIS

```c
#include <curses.h>

int insertln(void);
int winsertln(WINDOW *win);
```

PARAMETERS

- `win` Is a pointer to the window in which to insert the line.

DESCRIPTION

The `insertln()` and `winsertln()` functions insert a blank line before the current line in `stdscr` or `win`, respectively. The new line becomes the current line. The current line and all lines after it in the window are moved down one line. The bottom line in the window is discarded.

RETURN VALUES

On success, these functions return `OK`. Otherwise, they return `ERR`.

ERRORS

None.

SEE ALSO

`bkgdset(3XCURSES)`, `deleteln(3XCURSES)`, `insdelln(3XCURSES)`
NAME
insnstr, insstr, mvinsnstr, mvinsstr, mvwinsnstr, mvwinsstr, winsnstr, winsstr – insert a
multibyte character string

SYNOPSIS
#include <curses.h>

int insnstr(const char *str, int n);
int insstr(const char *str);
int mvinsnstr(int y, int x, const char *str, int n);
int mvinsstr(int y, int x, const char *str);
int mvwinsnstr(WINDOW *win, int y, int x, const char *str, int n);
int mvwinsstr(WINDOW *win, int y, int x, const char *str);
int winsnstr(WINDOW *win, const char *str, int n);
int winsstr(WINDOW *win, const char *str);

PARAMETERS
str Is a pointer to the string to be inserted.
n Is the number of characters not to exceed when
inserting str. If n is less than 1, the entire string is
inserted.
y Is the y (row) coordinate of the starting position of the
string.
x Is the x (column) coordinate of the starting position of the
string.
win Is a pointer to the window in which the string is to be
inserted.

DESCRIPTION
The insnstr() function inserts str at the current cursor position of the stdscr
window. The winsnstr() function performs the identical action, but in window win.
The mvinsnstr() and mvwinsstr() functions insert the character string at the
starting position indicated by the x (column) and y (row) parameters (the former to the
stdscr window; the latter to window win).

The insnstr(), winsnstr(), mvinsnstr(), and mvwinsstr() functions insert n
characters to the window or as many as will fit on the line. If n is less than 1, the
entire string is inserted or as much of it as fits on the line. The former two functions
place the string at the current cursor position; the latter two commands use the
position specified by the x and y parameters.

All characters to the right of inserted characters are moved to the right. Characters that
don’t fit on the current line are discarded. The cursor is left at the point of insertion.

If a character in str is a newline, carriage return, backspace, or tab, the cursor is moved
appropriately. The cursor is moved to the next tab stop for each tab character (by
default, tabs are eight characters apart). If the character is a control character other
than those previously mentioned, the character is inserted using \( ^x \) notation, where \( x \) is a printable character. `clrtoeol(3XCURSES)` is automatically done before a newline.

**RETURN VALUES**

On success, these functions return `OK`. Otherwise, they return `ERR`.

**ERRORS**

None.

**SEE ALSO**

`addchstr(3XCURSES)`, `addstr(3XCURSES)`, `clrtoeol(3XCURSES)`, `ins_nwstr(3XCURSES)`, `insch(3XCURSES)`
NAME

ins_nwstr, ins_wstr, mvins_nwstr, mvins_wstr, mvwins_nwstr, mvwins_nstr,
wins_nwstr, wins_wstr – insert a wide character string

SYNOPSIS

#include <curses.h>

int ins_nwstr(const wchar_t *wstr, int n);
int ins_wstr(const wchar_t *wstr);
int mvins_nwstr(int y, int x, const wchar_t *wstr, int n);
int mvins_wstr(int y, int x, const wchar_t *wstr);
int mvwins_nwstr(WINDOW *win, int y, int x, const wchar_t *wstr,
    int n);
int mvwins_wstr(WINDOW *win, int y, int x, const wchar_t *wstr);
int wins_nwstr(WINDOW *win, const wchar_t *wstr, int n);
int wins_wstr(WINDOW *win, const wchar_t *wstr);

PARAMETERS

wstr
  Is a pointer to the string to be inserted.

n
  Is the number of characters not to exceed when inserting wstr. If n is less than 1, the entire string is inserted.

y
  Is the y (row) coordinate of the starting position of the string.

x
  Is the x (column) coordinate of the starting position of the string.

win
  Is a pointer to the window in which the string is to be inserted.

DESCRIPTION

The ins_wstr() function inserts wstr at the current cursor position of the stdscr window. The wins_wstr() function performs the identical action, but in window win. The mvins_wstr() and mvwins_wstr() functions insert wstr string at the starting position indicated by the x (column) and y (row) parameters (the former in the stdscr window; the latter in window win).

The ins_nwstr(), wins_nwstr(), mvins_nwstr(), and mvwins_nwstr() functions insert n characters to the window or as many as will fit on the line. If n is less than 1, the entire string is inserted or as much of it as fits on the line. The former two functions place the string at the current cursor position; the latter two commands use the position specified by the x and y parameters.

All characters to the right of inserted characters are moved to the right. Characters that don’t fit on the current line are discarded. The cursor is left at the point of insertion.

If a character in wstr is a newline, carriage return, backspace, or tab, the cursor is moved appropriately. The cursor is moved to the next tab stop for each tab character.
ins_nwstr(3XCURSES)

(by default, tabs are eight characters apart). If the character is a control character other than those previously mentioned, the character is inserted using `^x notation, where x is a printable character. clrtoeol(3XCURSES) is automatically done before a newline.

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None.

SEE ALSO
add_wchnstr(3XCURSES), addnwstr(3XCURSES), clrtoeol(3XCURSES), ins_wch(3XCURSES), insnstr(3XCURSES)
ins_wch(3XCURSES)

NAME
ins_wch, wins_wch, mvins_wch, mvwins_wch – insert a complex character

SYNOPSIS
#include <curses.h>

int ins_wch(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);
int wins_wch(WINDOW *win, const cchar_t *wch);

PARAMETERS
wch
Is the complex character to be inserted.

y
Is the y (row) coordinate of the position of the character.

x
Is the x (column) coordinate of the position of the character.

win
Is a pointer to the window in which the character is to be inserted.

DESCRIPTION
The ins_wch() function inserts the complex character wch at the current cursor position of the stdscr window. The wins_wch() function performs the identical action but in window win. The mvins_wch() and mvwins_wch() functions insert the character at the position indicated by the x (column) and y (row) parameters (the former in the stdscr window; the latter in window win). The cursor position does not change.

All characters to the right of the inserted character are moved right one character. The last character on the line is deleted.

Insertions and deletions occur at the character level. The cursor is adjusted to the first column of the character prior to the operation.

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None.

SEE ALSO
add_wch(3XCURSES), ins_nwstr(3XCURSES)

254 man pages section 3: Curses Library Functions • Last Revised 1 Jun 1996
### intrflush(3XCURSES)

<table>
<thead>
<tr>
<th>NAME</th>
<th>intrflush – enable or disable flush on interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td><code>#include &lt;curses.h&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>int intrflush(WINDOW *win, bool bf);</code></td>
</tr>
<tr>
<td>PARAMETERS</td>
<td><code>win</code> Is ignored.</td>
</tr>
<tr>
<td></td>
<td><code>bf</code> Is a Boolean expression.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The <code>intrflush()</code> function specifies whether pressing an interrupt key (interrupt, suspend, or quit) will flush the input buffer associated with the current screen. If the value of <code>bf</code> is <code>TRUE</code>, then flushing of the output buffer associated with the current screen will occur when an interrupt key (interrupt, suspend, or quit) is pressed. If the value of <code>bf</code> is <code>FALSE</code>, then no flushing of the buffer will occur when an interrupt key is pressed. The default for the option is inherited from the display driver settings. The <code>win</code> argument is ignored.</td>
</tr>
<tr>
<td>RETURN VALUES</td>
<td>Upon successful completion, <code>intrflush()</code> returns <code>OK</code>. Otherwise, it returns <code>ERR</code>.</td>
</tr>
<tr>
<td>ERRORS</td>
<td>No errors are defined.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td><code>flushinp(3XCURSES), qiflush(3XCURSES)</code></td>
</tr>
</tbody>
</table>
in_wch(3XCURSES)

NAME  
in_wch, mvin_wch, mvwin_wch, win_wch – retrieve a complex character (with rendition)

SYNOPSIS  
#include <curses.h>

```c
int in_wch(cchar_t *wcval);
int mvin_wch(int y, int x, cchar_t *wcval);
int mvwin_wch(WINDOW *win, int y, cchar_t *wcval);
int win_wch(WINDOW *win, cchar_t *wcval);
```

DESCRIPTION  
The `in_wch()` and `win_wch()` functions retrieve the complex character and its rendition located at the current cursor position of the `stdscr` window and window `win`, respectively. The `mvin_wch()` and `mvwin_wch()` functions retrieve the complex character and its rendition located at the position indicated by the `x` (column) and `y` (row) parameters (the former in the `stdscr` window; the latter in window `win`).

All these functions store the retrieved character and its rendition in the object pointed to by `wcval`.

PARAMETERS  
- `wcval`  
  Is a pointer to an object that can store a complex character and its rendition.
- `y`  
  Is the y (row) coordinate of the position of the character to be returned.
- `x`  
  Is the x (column) coordinate of the position of the character to be returned.
- `win`  
  Is a pointer to the window that contains the character to be returned.

RETURN VALUES  
On success, these functions return `OK`. Otherwise, they return `ERR`.

ERRORS  
None.

SEE ALSO  
`add_wch(3XCURSES)`, `inch(3XCURSES)`
### NAME
in_wchnstr, in_wchstr, mvin_wchnstr, mvin_wchstr, mvwin_wchnstr, mvwin_wchstr,
win_wchnstr, win_wchstr – retrieve complex character string (with rendition)

### SYNOPSIS
```c
#include <curses.h>

int in_wchnstr(cchar_t *wchstr, int n);
int in_wchstr(cchar_t *wchstr);
int mvin_wchnstr(int y, int x, cchar_t *wchstr, int n);
int mvin_wchstr(int y, int x, cchar_t *wchstr);
int mvwin_wchnstr(WINDOW *win, int y, int x, cchar_t *wchstr, int n);
int mvwin_wchstr(WINDOW *win, int y, int x, cchar_t *wchstr);
int win_wchnstr(WINDOW *win, cchar_t *wchstr, int n);
int win_wchstr(WINDOW *win, cchar_t *wchstr);
```

### DESCRIPTION
The `in_wchnstr()` and `win_wchstr()` functions retrieve a complex character string
(with rendition) starting at the current cursor position of the `stdscr` window and
window `win`, respectively, and ending at the right margin. The `mvin_wchstr()` and
`mvwin_wchstr()` functions retrieve a complex character string located at the position
indicated by the `x` (column) and `y` (row) parameters (the former in the `stdscr`
window; the latter in window `win`).

The `in_wchnstr()`, `win_wchstr()`, `mvin_wchnstr()`, and `mvwin_wchstr()`
functions retrieve at most `n` characters from the window `stdscr` and `win`,
respectively. The former two functions retrieve the string, starting at the current cursor
position; the latter two commands retrieve the string, starting at the position specified
by the `x` and `y` parameters.

The retrieved character string (with renditions) is stored in the object pointed to by `wcval`.

### PARAMETERS
- **wchstr**
  Is a pointer to an object where the retrieved complex character string can be stored.
- **n**
  Is the number of characters not to exceed when retrieving `wchstr`.
- **y**
  Is the `y` (row) coordinate of the starting position of the string to be retrieved.
- **x**
  Is the `x` (column) coordinate of the starting position of the string to be retrieved.
- **win**
  Is a pointer to the window in which the string is to be retrieved.

### RETURN VALUES
On success, these functions return `OK`. Otherwise, they return `ERR`.

### ERRORS
None.
in_wchnstr(3XCURSES)

SEE ALSO in_wch(3XCURSES)
### NAME
is_linetouched, is_wintouched, touchline, touchwin, untouchwin, wtouchln – control window refresh

### SYNOPSIS
```
#include <curses.h>

bool is_linetouched(WINDOW *win, int line);
bool is_wintouchwin(WINDOW *win);
int touchline(WINDOW *win, int start, int count);
int touchwin(WINDOW *win);
int untouchwin(WINDOW *win);
int wtouchln(WINDOW *win, int y, int n, int changed);
```

### PARAMETERS
- `win` Is a pointer to the window in which the refresh is to be controlled or monitored.
- `line` Is the line to be checked for change since refresh.
- `start` Is the starting line number of the portion of the window to make appear changed.
- `count` Is the number of lines in the window to mark as changed.
- `y` Is the starting line number of the portion of the window to make appear changed or not changed.
- `n` Is the number of lines in the window to mark as changed.
- `changed` Is a flag indicating whether to make lines look changed (0) or not changed (1).

### DESCRIPTION
The `touchwin()` function marks the entire window as dirty. This makes it appear to X/Open Curses as if the whole window has been changed, thus causing the entire window to be rewritten with the next call to `refresh(3XCURSES)`. This is sometimes necessary when using overlapping windows; the change to one window will not be reflected in the other and, hence will not be recorded.

The `touchline()` function marks as dirty a portion of the window starting at line `start` and continuing for `count` lines instead of the entire window. Consequently, that portion of the window is updated with the next call to `refresh()`.

The `untouchwin()` function marks all lines in the window as unchanged since the last refresh, ensuring that it is not updated.

The `wtouchln()` function marks `n` lines starting at line `y` as either changed (`changed`=1) or unchanged (`changed`=0) since the last refresh.

To find out which lines or windows have been changed since the last refresh, use the `is_linetouched()` and `is_wintouched()` commands, respectively. These return
is_linetouched(3XCURSES)

TRUE if the specified line or window have been changed since the last call to
refresh() or FALSE if no changes have been made.

RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

ERRORS
None.

SEE ALSO
dupdate(3XCURSES)
NAME
keyname, key_name – return character string used as key name

SYNOPSIS
#include <curses.h>

char *keyname(int c);
char *key_name(wchar_t wc);

PARAMETERS
c Is an 8 bit-character or a key code.
wc Is a wide character key name.

DESCRIPTION
The keyname() function returns a string pointer to the key name. Make a duplicate copy of the returned string if you plan to modify it.

The key_name() function is similar except that it accepts a wide character key name.

The following table shows the format of the key name based on the input.

<table>
<thead>
<tr>
<th>Input</th>
<th>Format of Key Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible character</td>
<td>The same character</td>
</tr>
<tr>
<td>Control character</td>
<td>^X</td>
</tr>
<tr>
<td>Meta-character (keyname() only)</td>
<td>M-X</td>
</tr>
<tr>
<td>Key value defined in &lt;curses.h&gt;</td>
<td>KEY_name</td>
</tr>
<tr>
<td>None of the above</td>
<td>UNKNOWN KEY</td>
</tr>
</tbody>
</table>

In the preceding table, X can be either a visible character with the high bit cleared or a control character.

RETURN VALUES
On success, these functions return a pointer to the string used as the key’s name. Otherwise, they return a null pointer.

ERRORS
None.

SEE ALSO
meta(3XCURSES)
**NAME**  keypad – enable/disable keypad handling

**SYNOPSIS**
```
#include <curses.h>

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

**PARAMETERS**
- `win` Is a pointer to the window in which to enable/disable keypad handling.
- `bf` Is a Boolean expression.

**DESCRIPTION**
The `keypad()` function controls keypad translation. If `bf` is `TRUE`, keypad translation is enabled. If `bf` is `FALSE`, keypad translation is disabled. The initial state is `FALSE`.

This function affects the behavior of any function that provides keyboard input. If the terminal in use requires a command to enable it to transmit distinctive codes when a function key is pressed, then after keypad translation is first enabled, the implementation transmits this command to the terminal before an affected input function tries to read any characters from that terminal.

The Curses input model provides the following ways to obtain input from the keyboard:

**Keypad processing**
The application can enable or disable keypad translation by calling `keypad()`. When translation is enabled, Curses attempts to translate a sequence of terminal input that represents the pressing of a function into a single key code. When translation is disabled, Curses passes terminal input to the application without such translation, and any interpretation of the input as representing the pressing of a keypad key must be done by the application.

The complete set of key codes for keypad keys that Curses can process is specified by the constants defined in `<curses.h>` whose names begin with "KEY_". Each terminal type described in the `terminfo` database may support some or all of these key codes. The `terminfo` database specifies the sequence of input characters from the terminal type that correspond to each key code.

The Curses implementation cannot translate keypad keys on terminals where pressing the keys does not transmit a unique sequence.

When translation is enabled and a character that could be the beginning of a function key (such as `escape`) is received, Curses notes the time and begins accumulating characters. If Curses receives additional characters that represent the processing of a keypad key within an unspecified interval from the time the character was received, then Curses converts this input to a key code for presentation to the application. If such characters are not received during this interval, translation of this input does not occur and the individual characters are presented to the application separately. (Because Curses waits for this interval to accumulate a key code, many terminals experience a delay between the time a user presses the escape key and the time the escape key is returned to the application.)
In addition, No Timeout Mode provides that in any case where Curses has received part of a function key sequence, it waits indefinitely for the complete key sequence. The “unspecified interval” in the previous paragraph becomes infinite in No Timeout Mode. No Timeout Mode allows the use of function keys over slow communication lines. No Timeout Mode lets the user type the individual characters of a function key sequence, but also delays application response when the user types a character (not a function key) that begins a function key sequence. For this reason, in No Timeout Mode many terminals will appear to hang between the time a user presses the escape key and the time another key is pressed. No Timeout Mode is switchable by calling `notimeout(3XCURSES)`.

If any special characters (<backspace>, <carriage return>, <newline>, <tab>) are defined or redefined to be characters that are members of a function key sequence, then Curses will be unable to recognize and translate those function keys.

Several of the modes discussed below are described in terms of availability of input. If keypad translation is enabled, then input is not available once Curses has begun receiving a keypad sequence until the sequence is completely received or the interval has elapsed.

### Input Mode

The following four mutually-specific Curses modes let the application control the effect of flow-control characters, the interrupt character, the erase character, and the kill character:

<table>
<thead>
<tr>
<th>Input Mode</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cooked Mode</td>
<td>This achieves normal line-at-a-time processing with all special characters handled outside the application. This achieves the same effect as canonical-mode input processing. The state of the <code>ISIG</code> and <code>IXON</code> flags are not changed upon entering this mode by calling <code>nocbreak(3XCURSES)</code> and are set upon entering this mode by calling <code>noraw(3XCURSES)</code>. Erase and kill characters are supported from any supported locale, no matter the width of the character.</td>
</tr>
<tr>
<td>cbreak Mode</td>
<td>Characters typed by the user are immediately available to the application and Curses does not perform special processing on either the erase character or the kill character. An application can set <code>cbreak</code> mode to do its own line editing but to let the abort character be used to abort the task. This mode achieves the same effect as non-canonical-mode, Case B input processing (with <code>MIN</code> set to 1 and <code>ICRNL</code> cleared.) The state of the <code>ISIG</code> and <code>IXON</code> flags are not changed upon entering this mode.</td>
</tr>
</tbody>
</table>
### Input Mode Effect

<table>
<thead>
<tr>
<th>Input Mode</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Half-Delay Mode</td>
<td>The effect is the same as cbreak, except that input functions wait until a character is available or an interval defined by the application elapses, whichever comes first. This mode achieves the same effect as non-canonical-mode, Case C input processing (with TIME set to the value specified by the application.) The state of the ISIG and IXON flags are not changed upon entering this mode.</td>
</tr>
<tr>
<td>Raw Mode</td>
<td>Raw mode gives the application maximum control over terminal input. The application sees each character as it is typed. This achieves the same effect as non-canonical mode, Case D input processing. The ISIG and IXON flags are cleared upon entering this mode.</td>
</tr>
</tbody>
</table>

The terminal interface settings are reported when the process calls `initscr(3XCURSES)` or `newterm(3XCURSES)` to initialize Curses and restores these settings when `endwin(3XCURSES)` is called. The initial input mode for Curses operations is especially unless Enhanced Curses compliance, in which the initial mode is cbreak mode, is supported.

The behavior of the `BREAK` key depends on other bits in the display driver that are not set by Curses.

### Delay Mode

Two mutually-exclusive delay modes specify how quickly certain Curses functions return to the application when there is no terminal input waiting when the function is called:

<table>
<thead>
<tr>
<th>Delay Mode</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Delay</td>
<td>The function fails.</td>
</tr>
<tr>
<td>Delay</td>
<td>The application waits until text is passed through to the application. If cbreak or Raw Mode is set, this is after one character. Otherwise, this is after the first &lt;newline&gt; character, end-of-line character, or end-of-file character.</td>
</tr>
</tbody>
</table>

The effect of No Delay Mode on function key processing is unspecified.

### Echo processing

Echo mode determines whether Curses echoes typed characters to the screen. The effect of Echo mode is analogous to the effect of the ECHO flag in the local mode field of the `termios` structure associated with the terminal device connected to the window. However, Curses always clears the ECHO flag when invoked, to inhibit the operating system from performing echoing. The method of echoing characters is not identical to the operating system’s method of echoing characters, because Curses performs additional processing of terminal input.

If in Echo mode, Curses performs ‘s’s own echoing. Any visible input character is stored in the current or specified window by the input function that the application called, at that window’s cursor position, as though `addch(3XCURSES)` were called, with all consequent effects such as cursor movement and wrapping.
If not in Echo mode, any echoing of input must be performed by the application. Applications often perform their own echoing in a controlled area of the screen, or do not echo at all, so they disable Echo mode.

It may not be possible to turn off echo processing for synchronous and networked asynchronous terminals because echo processing is done directly by the terminals. Applications running on such terminals should be aware that any characters typed will appear on the screen at wherever the cursor is positioned.

RETURN VALUES
Upon successful completion, the `keypad()` function returns `OK`. Otherwise, it returns `ERR`.

ERRORS
No errors are defined.

SEE ALSO
`addch(3XCURSES)`, `endwin(3XCURSES)`, `getch(3XCURSES)`, `initscr(3XCURSES)`, `newterm(3XCURSES)`, `nocbreak(3XCURSES)`, `noraw(3XCURSES)`
<table>
<thead>
<tr>
<th>NAME</th>
<th>LINES – number of lines on terminal screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>#include &lt;curses.h&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>extern int LINES;</code></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The external variable LINES indicates the number of lines on the terminal screen.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td><code>initscr(3XCURSES)</code></td>
</tr>
</tbody>
</table>
NAME

longname – return full terminal type name

SYNOPSIS

#include <curses.h>

const char *longname(void);

DESCRIPTION

The longname() function returns a pointer to a static area containing a verbose
description (128 characters or fewer) of the terminal. The area is defined after calls to
initscr(3XCURSES), newterm(3XCURSES), or setupterm(3XCURSES). The value
should be saved if longname() is going to be used with multiple terminals since it
will be overwritten with a new value after each call to newterm() or setupterm().

RETURN VALUES

On success, the longname() function returns a pointer to a verbose description of the
terminal. Otherwise, it returns a null pointer.

ERRORS

None.

SEE ALSO

initscr(3XCURSES), newterm(3XCURSES), setupterm(3XCURSES)
NAME

menu_attributes, set_menu_fore, menu_fore, set_menu_back, menu_back,
set_menu_grey, menu_grey, set_menu_pad, menu_pad – control menus display
attributes

SYNOPSIS

cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>

int set_menu_fore(MENU *menu, chtype attr);
chtype menu_fore(MENU *menu);
int set_menu_back(MENU *menu, chtype attr);
chtype menu_back(MENU *menu);
int set_menu_grey(MENU*menu, chtype attr);
chtype menu_grey(MENU *menu);
int set_menu_pad(MENU *menu, int pad);
int menu_pad(MENU *menu);

DESCRIPTION

set_menu_fore() sets the foreground attribute of menu — the display attribute for
the current item (if selectable) on single-valued menus and for selected items on
multi-valued menus. This display attribute is a curses library visual attribute.

.menu_fore() returns the foreground attribute of menu.

set_menu_back() sets the background attribute of menu — the display attribute for
unselected, yet selectable, items. This display attribute is a curses library visual
attribute.

set_menu_grey() sets the grey attribute of menu — the display attribute for
nonselectable items in multi-valued menus. This display attribute is a curses library
visual attribute. menu_grey() returns the grey attribute of menu.

The pad character is the character that fills the space between the name and
description of an item. set_menu_pad() sets the pad character for menu to pad.
menu_pad() returns the pad character of menu.

RETURN VALUES

These routines return one of the following:

E_OK The routine returned successfully.
E_SYSTEM_ERROR System error.
E_BAD_ARGUMENT An incorrect argument was passed to the routine.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>
The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

**SEE ALSO**
curses(3CURSES), menus(3CURSES), attributes(5)

**NOTES**
The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.
menu_cursor(3CURSES)

NAME
menu_cursor, pos_menu_cursor - correctly position a menus cursor

SYNOPSIS
ec [ flag ... ] file ... -lmenu -lcurses [ library .. ]
#include <menu.h>

int pos_menu_cursor(MENU *menu);

DESCRIPTION
pos_menu_cursor() moves the cursor in the window of menu to the correct position
to resume menu processing. This is needed after the application calls a curses library
I/O routine.

RETURN VALUES
This routine returns one of the following:
E_OK The routine returned successfully.
E_SYSTEM_ERROR System error.
E_BAD_ARGUMENT An incorrect argument was passed to the routine.
E_NOT_POSTED The menu has not been posted.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), menus(3CURSES), panel_update(3CURSES),
panels(3CURSES), attributes(5)

NOTES
The header <menu.h> automatically includes the headers <eti.h> and
<curses.h>.
NAME

menu_driver – command processor for the menus subsystem

SYNOPSIS

cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>

int menu_driver(MENU *menu, int c);

DESCRIPTION

menu_driver() is the workhorse of the menus subsystem. It checks to determine whether the character c is a menu request or data. If c is a request, the menu driver executes the request and reports the result. If c is data (a printable ASCII character), it enters the data into the pattern buffer and tries to find a matching item. If no match is found, the menu driver deletes the character from the pattern buffer and returns E_NO_MATCH. If the character is not recognized, the menu driver assumes it is an application-defined command and returns E_UNKNOWN_COMMAND.

Menu driver requests:

- REQ_LEFT_ITEM: Move left to an item.
- REQ_RIGHT_ITEM: Move right to an item.
- REQ_UP_ITEM: Move up to an item.
- REQ_DOWN_ITEM: Move down to an item.
- REQ_SCR_U_LINE: Scroll up a line.
- REQ_SCR_D_LINE: Scroll down a line.
- REQ_SCR_D_PAGE: Scroll up a page.
- REQ_SCR_U_PAGE: Scroll down a page.
- REQ_FIRST_ITEM: Move to the first item.
- REQ_LAST_ITEM: Move to the last item.
- REQ_NEXT_ITEM: Move to the next item.
- REQ_PREV_ITEM: Move to the previous item.
- REQ_TOGGLE_ITEM: Select/de-select an item.
- REQ_CLEAR_PATTERN: Clear the menu pattern buffer.
- REQ_BACK_PATTERN: Delete the previous character from pattern buffer.
- REQ_NEXT_MATCH: Move the next matching item.
- REQ_PREV_MATCH: Move to the previous matching item.

RETURN VALUES

menu_driver() returns one of the following:

- E_OK: The routine returned successfully.
- E_SYSTEM_ERROR: System error.
E_BAD_ARGUMENT An incorrect argument was passed to the routine.

E_BAD_STATE The routine was called from an initialization or termination function.

E_NOT_POSTED The menu has not been posted.

E_UNKNOWN_COMMAND An unknown request was passed to the menu driver.

E_NO_MATCH The character failed to match.

E_NOT_SELECTABLE The item cannot be selected.

E_REQUEST_DENIED The menu driver could not process the request.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3CURSES), menus(3CURSES), attributes(5)

NOTES Application defined commands should be defined relative to (greater than) MAX_COMMAND, the maximum value of a request listed above.

The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
menu_format(3CURSES)

NAME  menu_format, set_menu_format – set and get maximum numbers of rows and columns in menus

SYNOPSIS  cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>

int set_menu_format(MENU *menu, int rows, int cols);
void menu_format(MENU *menu, int *rows, int *cols);

DESCRIPTION  set_menu_format() sets the maximum number of rows and columns of items that may be displayed at one time on a menu. If the menu contains more items than can be displayed at once, the menu will be scrollable.

menu_format() returns the maximum number of rows and columns that may be displayed at one time on menu. rows and cols are pointers to the variables used to return these values.

RETURN VALUES  set_menu_format() returns one of the following:
E_OK The routine returned successfully.
E_SYSTEM_ERROR System error.
E_BAD_ARGUMENT An incorrect argument was passed to the routine.
E_POSTED The menu is already posted.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3CURSES), menus(3CURSES), attributes(5)

NOTES  The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
menu_hook(3CURSES)

NAME
menu_hook, set_item_init, item_init, set_item_term, item_term, set_menu_init,
menu_init, set_menu_term, menu_term - assign application-specific routines for
automatic invocation by menus

SYNOPSIS
cc [... flags ...] file ... -lmencurses [ library ...]
#include <menu.h>

int set_item_init(MENU *menu, void (*func)(MENU *));
void (*item_init)(MENU *menu);
int set_item_term(MENU *menu, void (*func)(MENU *));
void (*item_term)(MENU *menu);
int set_menu_init(MENU *menu, void (*func)(MENU *));
void (*menu_init)(MENU *menu);
int set_menu_term(MENU *menu, void (*func)(MENU *));
void (*menu_term)(MENU *menu);

DESCRIPTION
set_item_init() assigns the application-defined function to be called when the
menu is posted and just after the current item changes. item_init() returns a
pointer to the item initialization routine, if any, called when the menu is posted and
just after the current item changes.

set_item_term() assigns an application-defined function to be called when the
menu is unposted and just before the current item changes. item_term() returns a
pointer to the termination function, if any, called when the menu is unposted and just
before the current item changes.

set_menu_init() assigns an application-defined function to be called when the
menu is posted and just after the top row changes on a posted menu. menu_init() returns a
pointer to the menu initialization routine, if any, called when the menu is posted and just
after the top row changes on a posted menu.

set_menu_term() assigns an application-defined function to be called when the
menu is unposted and just before the top row changes on a posted menu.
menu_term() returns a pointer to the menu termination routine, if any, called when
the menu is unposted and just before the top row changes on a posted menu.

RETURN VALUES
Routines that return pointers always return NULL on error. Routines that return an
integer return one of the following:
E_OK The routine returned successfully.
E_SYSTEM_ERROR System error.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:
The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

**SEE ALSO**
curses(3CURSES), menus(3CURSES), attributes(5)

**NOTES**
The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>
NAME

menu_item_current, set_current_item, current_item, set_top_row, top_row, item_index

set and get current menus items

SYNOPSIS

cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]

#include <menu.h>

int set_current_item(MENU *menu, ITEM *item);
ITEM *current_item(MENU *menu);
int set_top_row(MENU *menu, int row);
int top_row(MENU *menu);
int item_index(ITEM *item);

DESCRIPTION

The current item of a menu is the item where the cursor is currently positioned.

set_current_item() sets the current item of menu to item. current_item() returns a pointer to the the current item in menu.

set_top_row() sets the top row of menu to row. The left-most item on the new top row becomes the current item. top_row() returns the number of the menu row currently displayed at the top of menu.

item_index() returns the index to the item in the item pointer array. The value of this index ranges from 0 through N-1, where N is the total number of items connected to the menu.

RETURN VALUES

current_item() returns NULL on error.

top_row() and index_item() return -1 on error.

set_current_item() and set_top_row() return one of the following:

E_OK The routine returned successfully.
E_SYSTEM_ERROR System error.
E_BAD_ARGUMENT An incorrect argument was passed to the routine.
E_BAD_STATE The routine was called from an initialization or termination function.
E_NOT_CONNECTED No items are connected to the menu.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3CURSES), menus(3CURSES), attributes(5)
The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
menu_item_name(3CURSES)

NAME menu_item_name, item_name, item_description – get menus item name and description

SYNOPSIS cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>

char *item_name(ITEM *item);
char *item_description(ITEM *item);

DESCRIPTION item_name() returns a pointer to the name of item.

item_description() returns a pointer to the description of item.

RETURN VALUES These routines return NULL on error.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3CURSES), menus(3CURSES), menu_new(3CURSES), attributes(5)

NOTES The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.

278 man pages section 3: Curses Library Functions • Last Revised 31 Dec 1996
menu_item_new(3CURSES)

NAME menu_item_new, new_item, free_item – create and destroy menus items

SYNOPSIS cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>

ITEM *new_item(char *name, char *desc);
int free_item(ITEM *item);

DESCRIPTION new_item() creates a new item from name and description, and returns a pointer to the new item.

free_item() frees the storage allocated for item. Once an item is freed, the user can no longer connect it to a menu.

RETURN VALUES new_item() returns NULL on error.

free_item() returns one of the following:
  E_OK The routine returned successfully.
  E_SYSTEM_ERROR System error.
  E_BAD_ARGUMENT An incorrect argument was passed to the routine.
  E_CONNECTED One or more items are already connected to another menu.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3CURSES), menus(3CURSES), attributes(5)

NOTES The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME menu_item_opts(3CURSES)

SYNOPSIS cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>

int set_item_opts(ITEM *item, OPTIONS opts);
int item_opts_on(ITEM *item, OPTIONS opts);
int item_opts_off(ITEM *item, OPTIONS opts);
OPTIONS item_opts(ITEM *item);

DESCRIPTION set_item_opts() turns on the named options for item and turns off all other options. Options are boolean values that can be OR-ed together.

item_opts_on() turns on the named options for item; no other option is changed.

item_opts_off() turns off the named options for item; no other option is changed.

item_opts() returns the current options of item.

O_SELECTABLE The item can be selected during menu processing.

RETURN VALUES Except for item_opts(), these routines return one of the following:

E_OK The routine returned successfully.

E_SYSTEM_ERROR System error.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3CURSES), menus(3CURSES), attributes(5)

NOTES The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
menu_items(3CURSES)

NAME menu_items, set_menu_items, item_count – connect and disconnect items to and from menus

SYNOPSIS cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>

int set_menu_items(MENU *menu, ITEM **items);
ITEM **menu_items(MENU *menu);
int item_count(MENU *menu);

DESCRIPTION set_menu_items() changes the item pointer array connected to menu to the item pointer array items. menu_items() returns a pointer to the item pointer array connected to menu. item_count() returns the number of items in menu.

RETURN VALUES menu_items() returns NULL on error.
item_count() returns -1 on error.
set_menu_items() returns one of the following:
E_OK The routine returned successfully.
E_SYSTEM_ERROR System error.
E_BAD_ARGUMENT An incorrect argument was passed to the routine.
E_POSTED The menu is already posted.
E_CONNECTED One or more items are already connected to another menu.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO curses(3CURSES), menus(3CURSES), attributes(5)

NOTES The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.

Curses Library Functions 281
NAME
menu_item_userptr(3CURSES)

SYNOPSIS
cc [ flag ... ] file ... -lmenu -lcurses [ library .. ]
#include <menu.h>

int set_item_userptr(ITEM *item, char *userptr);

char *item_userptr(ITEM *item);

DESCRIPTION
Every item has an associated user pointer that can be used to store relevant
information. set_item_userptr() sets the user pointer of item. item_userptr() returns the user pointer of item.

RETURN VALUES
item_userptr() returns NULL on error. set_item_userptr() returns one of the
following:
E_OK The routine returned successfully.
E_SYSTEM_ERROR System error.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), menus(3CURSES), attributes(5)

NOTES
The header <menu.h> automatically includes the headers <eti.h> and
<curses.h>.
### NAME
menu_item_value, set_item_value, item_value – set and get menus item values

### SYNOPSIS
```c
#include <menu.h>

int set_item_value(ITEM *item, int bool);
int item_value(ITEM *item);
```

### DESCRIPTION
Unlike single-valued menus, multi-valued menus enable the end-user to select one or more items from a menu. set_item_value() sets the selected value of the item — TRUE (selected) or FALSE (not selected). set_item_value() may be used only with multi-valued menus. To make a menu multi-valued, use set_menu_opts or menu_opts_off() to turn off the option O_ONEVALUE. (See menu_opts(3CURSES)).

item_value() returns the select value of item, either TRUE (selected) or FALSE (unselected).

### RETURN VALUES
set_item_value() returns one of the following:
- E_OK The routine returned successfully.
- E_SYSTEM_ERROR System error.
- E_REQUEST_DENIED The menu driver could not process the request.

### ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

### SEE ALSO
curses(3CURSES), menus(3CURSES), menu_opts(3CURSES), attributes(5)

### NOTES
The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
**NAME**

`menu_item_visible`, `item_visible` – tell if menus item is visible

**SYNOPSIS**

```c
#include <menu.h>

int item_visible(ITEM *item);
```

**DESCRIPTION**

A menu item is visible if it currently appears in the subwindow of a posted menu. `item_visible()` returns `TRUE` if `item` is visible, otherwise it returns `FALSE`.

**ATTRIBUTES**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`curses(3CURSES)`, `menus(3CURSES)`, `menu_new(3CURSES)`, `attributes(5)`

**NOTES**

The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.
NAME  menu_mark, set_menu_mark – menus mark string routines

SYNOPSIS  cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
           #include <menu.h>

           int set_menu_mark(MENU *menu, char *mark);
           char *menu_mark(MENU *menu);

DESCRIPTION  menus displays mark strings to distinguish selected items in a menu (or the current
              item in a single-valued menu). set_menu_mark() sets the mark string of menu to
              mark. menu_mark() returns a pointer to the mark string of menu.

RETURN VALUES  menu_mark() returns NULL on error. set_menu_mark() returns one of the
              following:
              E_OK The routine returned successfully.
              E_SYSTEM_ERROR System error.
              E_BAD_ARGUMENT An incorrect argument was passed to the routine.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3CURSES), menus(3CURSES), attributes(5)

NOTES  The header <menu.h> automatically includes the headers <eti.h> and
        <curses.h>.
menu_new(3CURSES)

NAME    menu_new, new_menu, free_menu – create and destroy menus

SYNOPSIS   cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
            #include <menu.h>

            MENU *new_menu(ITEM **items);
            int free_menu(MENU *menu);

DESCRIPTION  new_menu() creates a new menu connected to the item pointer array items
              and returns a pointer to the new menu.

            free_menu() disconnects menu from its associated item pointer array
            and frees the storage allocated for the menu.

RETURN VALUES  new_menu() returns NULL on error.

            free_menu() returns one of the following:

              E_OK         The routine returned successfully.
              E_SYSTEM_ERROR   System error.
              E_BAD_ARGUMENT   An incorrect argument was passed to the routine.
              E_POSTED       The menu is already posted.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:


<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3CURSES), menus(3CURSES), attributes(5)

NOTES  The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME

menu_opts, set_menu_opts, menu_opts_on, menu_opts_off – menus option routines

SYNOPSIS

cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>

OPTIONS

int menu_opts(MENU *menu);
int set_menu_opts(MENU *menu, OPTIONS opts);
int menu_opts_on(MENU *menu, OPTIONS opts);
int menu_opts_off(MENU *menu, OPTIONS opts);

Menu Options

set_menu_opts() turns on the named options for menu and turns off all other options. Options are boolean values that can be OR-ed together.

menu_opts_on() turns on the named options for menu; no other option is changed.

menu_opts_off() turns off the named options for menu; no other option is changed.

menu_opts() returns the current options of menu.

The following values can be OR’d together to create opts.

O_ONEVALUE  Only one item can be selected from the menu.
O_SHOWDESC  Display the description of the items.
O_ROWMAJOR  Display the menu in row major order.
O_IGNORECASE Ignore the case when pattern matching.
O_SHOWMATCH Place the cursor within the item name when pattern matching.
O_NONCYCLIC Make certain menu driver requests non-cyclic.

RETURN VALUES

Except for menu_opts(), these routines return one of the following:

E_OK  The routine returned successfully.
E_SYSTEM_ERROR  System error.
E_POSTED  The menu is already posted.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

curses(3CURSES), menus(3CURSES), attributes(5)

NOTES

The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
menu_pattern(3CURSES)

NAME  menu_pattern, set_menu_pattern – set and get menus pattern match buffer

SYNOPSIS  cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>

char *menu_pattern(MENU *menu);

int set_menu_pattern(MENU *menu, char *pat);

DESCRIPTION  Every menu has a pattern buffer to match entered data with menu items.
set_menu_pattern() sets the pattern buffer to pat and tries to find the first item
that matches the pattern. If it does, the matching item becomes the current item. If not,
the current item does not change. menu_pattern() returns the string in the pattern
buffer of menu.

RETURN VALUES  menu_pattern() returns NULL on error. set_menu_pattern() returns one of the
following:
E_OK The routine returned successfully.
E_SYSTEM_ERROR System error.
E_BAD_ARGUMENT An incorrect argument was passed to the routine.
E_NO_MATCH The character failed to match.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3CURSES), menus(3CURSES), attributes(5)

NOTES  The header <menu.h> automatically includes the headers <eti.h> and
<curses.h>.
NAME

menu_post, post_menu, unpost_menu – write or erase menus from associated subwindows

SYNOPSIS

```c
cc [ flag ... ] file ... -lmenu -lcurses [ library ... ]
#include <menu.h>

int post_menu(MENU *menu);
int unpost_menu(MENU *menu);
```

DESCRIPTION

post_menu() writes menu to the subwindow. The application programmer must use curses library routines to display the menu on the physical screen or call update_panels() if the panels library is being used.

unpost_menu() erases menu from its associated subwindow.

RETURN VALUES

These routines return one of the following:

- **E_OK** The routine returned successfully.
- **E_SYSTEM_ERROR** System error.
- **E_BAD_ARGUMENT** An incorrect argument was passed to the routine.
- **E_POSTED** The menu is already posted.
- **E_BAD_STATE** The routine was called from an initialization or termination function.
- **E_NO_ROOM** The menu does not fit within its subwindow.
- **E_NOT_POSTED** The menu has not been posted.
- **E_NOT_CONNECTED** No items are connected to the menu.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

curses(3CURSES), menus(3CURSES), panels(3CURSES), attributes(5)

NOTES

The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.
menus(3CURSES)

NAME
menus – character based menus package

SYNOPSIS
#include <menu.h>

DESCRIPTION
The menu library is built using the curses library, and any program using menus routines must call one of the curses initialization routines, such as initscr. A program using these routines must be compiled with -lmenu and -lcurses on the cc command line.

The menus package gives the applications programmer a terminal-independent method of creating and customizing menus for user interaction. The menus package includes: item routines, which are used to create and customize menu items; and menu routines, which are used to create and customize menus, assign pre- and post-processing routines, and display and interact with menus.

The menus package establishes initial current default values for item attributes. During item initialization, each item attribute is assigned the current default value for that attribute. An application can change or retrieve a current default attribute value by calling the appropriate set or retrieve routine with a NULL item pointer. If an application changes a current default item attribute value, subsequent items created using new_item() will have the new default attribute value. The attributes of previously created items are not changed if a current default attribute value is changed.

The following table lists each menus routine and the name of the manual page on which it is described.

<table>
<thead>
<tr>
<th>Menus Routine Name</th>
<th>Manual Page Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_item</td>
<td>menu_item_current(3X)</td>
</tr>
<tr>
<td>free_item</td>
<td>menu_item_new(3X)</td>
</tr>
<tr>
<td>free_menu</td>
<td>menu_new(3X)</td>
</tr>
<tr>
<td>item_count</td>
<td>menu_items(3X)</td>
</tr>
<tr>
<td>item_description</td>
<td>menu_item_name(3X)</td>
</tr>
<tr>
<td>item_index</td>
<td>menu_item_current(3X)</td>
</tr>
<tr>
<td>item_init</td>
<td>menu_hook(3X)</td>
</tr>
<tr>
<td>item_name</td>
<td>menu_item_name(3X)</td>
</tr>
<tr>
<td>item_opts</td>
<td>menu_item_opts(3X)</td>
</tr>
<tr>
<td>item_opts_off</td>
<td>menu_item_opts(3X)</td>
</tr>
<tr>
<td>item_opts_on</td>
<td>menu_item_opts(3X)</td>
</tr>
<tr>
<td>item_term</td>
<td>menu_hook(3X)</td>
</tr>
</tbody>
</table>

Current Default Values for Item Attributes

Routine Name Index

290 man pages section 3: Curses Library Functions • Last Revised 31 Dec 1996
# menus(3CURSES)

<table>
<thead>
<tr>
<th>Menu Routine Name</th>
<th>Manual Page Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>item_userptr</td>
<td>menu_item_userptr(3X)</td>
</tr>
<tr>
<td>item_value</td>
<td>menu_item_value(3X)</td>
</tr>
<tr>
<td>item_visible</td>
<td>menu_item_visible(3X)</td>
</tr>
<tr>
<td>menu_back</td>
<td>menu_attributes(3X)</td>
</tr>
<tr>
<td>menu_driver</td>
<td>menu_driver(3X)</td>
</tr>
<tr>
<td>menu_fore</td>
<td>menu_attributes(3X)</td>
</tr>
<tr>
<td>menu_format</td>
<td>menu_format(3X)</td>
</tr>
<tr>
<td>menu_grey</td>
<td>menu_attributes(3X)</td>
</tr>
<tr>
<td>menu_init</td>
<td>menu_hook(3X)</td>
</tr>
<tr>
<td>menu_items</td>
<td>menu_items(3X)</td>
</tr>
<tr>
<td>menu_mark</td>
<td>menu_mark(3X)</td>
</tr>
<tr>
<td>menu_opts</td>
<td>menu_opts(3X)</td>
</tr>
<tr>
<td>menu_opts_off</td>
<td>menu_opts(3X)</td>
</tr>
<tr>
<td>menu_opts_on</td>
<td>menu_opts(3X)</td>
</tr>
<tr>
<td>menu_pad</td>
<td>menu_attributes(3X)</td>
</tr>
<tr>
<td>menu_pattern</td>
<td>menu_pattern(3X)</td>
</tr>
<tr>
<td>menu_sub</td>
<td>menu_win(3X)</td>
</tr>
<tr>
<td>menu_term</td>
<td>menu_hook(3X)</td>
</tr>
<tr>
<td>menu_userptr</td>
<td>menu_userptr(3X)</td>
</tr>
<tr>
<td>menu_win</td>
<td>menu_win(3X)</td>
</tr>
<tr>
<td>new_item</td>
<td>menu_item_new(3X)</td>
</tr>
<tr>
<td>new_menu</td>
<td>menu_new(3X)</td>
</tr>
<tr>
<td>pos_menu_cursor</td>
<td>menu_cursor(3X)</td>
</tr>
<tr>
<td>post_menu</td>
<td>menu_post(3X)</td>
</tr>
<tr>
<td>scale_menu</td>
<td>menu_win(3X)</td>
</tr>
<tr>
<td>set_current_item</td>
<td>menu_item_current(3X)</td>
</tr>
<tr>
<td>set_item_init</td>
<td>menu_hook(3X)</td>
</tr>
<tr>
<td>set_item_opts</td>
<td>menu_item_opts(3X)</td>
</tr>
<tr>
<td>set_item_term</td>
<td>menu_hook(3X)</td>
</tr>
</tbody>
</table>
Routines that return pointers always return NULL on error. Routines that return an integer return one of the following:

- **E_OK**: The routine returned successfully.
- **E_SYSTEM_ERROR**: System error.
- **E_BAD_ARGUMENT**: An incorrect argument was passed to the routine.
- **E_POSTED**: The menu is already posted.
- **E_CONNECTED**: One or more items are already connected to another menu.
- **E_BAD_STATE**: The routine was called from an initialization or termination function.
The menu does not fit within its subwindow.
The menu has not been posted.
An unknown request was passed to the menu driver.
The character failed to match.
The item cannot be selected.
No items are connected to the menu.
The menu driver could not process the request.
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

See also curses(3CURSES), attributes(5)

The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
menu_userptr(3CURSES)

NAME  menu_userptr, set_menu_userptr – associate application data with menus

SYNOPSIS  cc [ flag ... ] file ... -lmenu -lcurses [ library .. ]
#include <menu.h>

char *menu_userptr(MENU *menu);

int set_menu_userptr(MENU *menu, char *userptr);

DESCRIPTION  Every menu has an associated user pointer that can be used to store relevant
information. set_menu_userptr() sets the user pointer of menu. menu_userptr() returns the user pointer of menu.

RETURN VALUES  menu_userptr() returns NULL on error.

set_menu_userptr() returns one of the following:

E_OK  The routine returned successfully.

E_SYSTEM_ERROR  System error.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3CURSES), menus(3CURSES), attributes(5)

NOTES  The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
NAME

menu_win, set_menu_win, set_menu_sub, menu_sub, scale_menu – menus window and subwindow association routines

SYNOPSIS

cc [ flag ... ] file ... -lmenu -lcurses [ library .. ]
#include <menu.h>

int set_menu_win(MENU *, WINDOW *win);
WINDOW *menu_win(MENU *menu);
int set_menu_sub(MENU *, WINDOW *sub);
WINDOW *menu_sub(MENU *menu);
int scale_window(MENU *, int *, int *rows, int *cols);

DESCRIPTION

set_menu_win() sets the window of menu to win. menu_win() returns a pointer to the window of menu. set_menu_sub() sets the subwindow of menu to sub. menu_sub() returns a pointer to the subwindow of menu. scale_window() returns the minimum window size necessary for the subwindow of menu. rows and cols are pointers to the locations used to return the values.

RETURN VALUES

Routines that return pointers always return NULL on error. Routines that return an integer return one of the following:

E_OK The routine returned successfully.
E_SYSTEM_ERROR System error.
E_BAD_ARGUMENT An incorrect argument was passed to the routine.
E_POSTED The menu is already posted.
E_NOT_CONNECTED No items are connected to the menu.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

curses(3CURSES), menus(3CURSES), attributes(5)

NOTES

The header <menu.h> automatically includes the headers <eti.h> and <curses.h>.
### NAME
meta – enable/disable meta keys

### SYNOPSIS
```c
#include <curses.h>

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

### PARAMETERS
- **win**
  Is an ignored parameter.
- **bf**
  Is a Boolean expression.

### DESCRIPTION
Whether a terminal returns 7 or 8 significant bits initially depends on the control mode of the terminal driver. The `meta()` function forces the number of bits to be returned by `getch(3XCURSES)` to be 7 (if `bf` is `FALSE`) or 8 (if `bf` is `TRUE`).

If the program handling the data can only pass 7-bit characters or strips the 8th bit, 8 bits cannot be handled.

If the `terminfo` capabilities `smm` (meta_on) and `rmm` (meta_off) are defined for the terminal, `smm` is sent to the terminal when `meta(win, TRUE)` is called, and `rmm` is sent when `meta(win, FALSE)` is called.

This function is useful when extending the non-text command set in applications where the META key is used.

### RETURN VALUES
On success, the `meta()` function returns `OK`. Otherwise, it returns `ERR`.

### ERRORS
None.

### SEE ALSO
`getch(3XCURSES)`
#NAME
move, wmove – move cursor in window

#SYNOPSIS
#include <curses.h>

int move(int y, int x);

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

#PARAMETERS
y
Is the y (row) coordinate of the position of the cursor in the window.

x
Is the x (column) coordinate of the position of the cursor in the window.

win
Is a pointer to the window in which the cursor is to be written.

#DESCRIPTION
The move() function moves the logical cursor (for stdscr) to the position specified by y (row) and x (column), where the upper left corner of the window is row 0, column 0. The wmove() function performs the same action, but moves the cursor in the window specified by win. The physical cursor will not move until after a call to refresh(3XCURSES) or doupdate(3XCURSES).

#RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

#ERRORS
None.

#SEE ALSO
doupdate(3XCURSES)
mvcur(3XCURSES)

NAME  mvcur – move the cursor

SYNOPSIS  
```c
#include <curses.h>

int mvcur(int oldrow, int oldcol, int newrow, int newcol);
```

PARAMETERS  
- `oldrow`  Is the row from which cursor is to be moved.
- `oldcol`  Is the column from which cursor is to be moved.
- `newrow`  Is the row to which cursor is to be moved.
- `newcol`  Is the column to which cursor is to be moved.

DESCRIPTION  
The `mvcur()` function is a low-level function used only outside of X/Open Curses when the program has to deal directly with the `terminfo` database to handle certain terminal capabilities. The use of appropriate X/Open Curses functions is recommended in all other situations, so that X/Open Curses can track the cursor.

The `mvcur()` function moves the cursor from the location specified by `oldrow` and `oldcol` to the location specified by `newrow` and `newcol`. A program using this function must keep track of the current cursor position.

RETURN VALUES  
On success, the `mvcur()` function returns `OK`. Otherwise, it returns `ERR`.

ERRORS  
None.
mvderwin() – map area of parent window to subwindow

SYNOPSIS

```
#include <curses.h>

int mvderwin(WINDOW *win, int par_y, int par_x);
```

PARAMETERS

- **win**: Is a pointer to the window to be mapped.
- **par_y**: Is the y (row) coordinate of the placement of the upper left corner of window relative to the parent window.
- **par_x**: Is the x (column) coordinate of the placement of the upper left corner of the window relative to the parent window.

DESCRIPTION

The `mvderwin()` function defines a mapped area of `win`'s parent window that is the same size as `win` and has its upper left corner at position `par_y, par_x` of the parent window.

Whenever `win` is refreshed, its contents are updated to match those of the mapped area and any reference to characters in `win` is treated as a reference to corresponding characters in the mapped area.

RETURN VALUES

On success, the `mvderwin()` function returns `OK`. Otherwise, it returns `ERR`.

ERRORS

None.

SEE ALSO

`delwin(3CURSES), derwin(3CURSES)`
mvprintw(3XCURSES)

NAME       mvprintw, mvwprintw, printw, wprintw – print formatted output window

SYNOPSIS   #include <curses.h>

int mvprintw(int y, int x, char *fmt, ...);
int mvwprintw(WINDOW *win, int y, int x, char *fmt, ...);
int printw(char *fmt, ...);
int wprintw(WINDOW *win, char *fmt, ...);

PARAMETERS y    Is the y (row) coordinate position of the string’s placement in the window.
 x    Is the x (column) coordinate position of the string’s placement in the window.
 fmt   Is a printf() format string.
 win   Is a pointer to the window in which the string is to be written.

DESCRIPTION The mvprintw(), mvwprintw(), printw(), and wprintw() functions are analogous to printf(3C). The effect of these functions is as though sprintf() were used to format the string, and then waddstr(3XCURSES) were used to add that multi-byte string to the current or specified window at the current or specified cursor position.

RETURN VALUES Upon successful completion, these functions return OK. Otherwise, they return ERR.

ERRORS No errors are defined.

SEE ALSO addnstr(3XCURSES), printf(3C)
mvscanw, mvwscanw, scanw, wscanw – convert formatted input from a window

### Synopsis

```c
#include <curses.h>

int mvscanw(int y, int x, char *fmt, ...);
int mvwscanw(WINDOW *win, int y, int x, char *fmt, ...);
int scanw(char *fmt, ...);
int wscanw(WINDOW *win, char *fmt, ...);
```

### Parameters

- **y**: Is the y (row) coordinate of the position of the character to be read.
- **x**: Is the x (column) coordinate of the position of the character to be read.
- **fmt**: Is a `scanf()` format string.
- **win**: Is a pointer to the window in which the character is to be read.

### Description

These functions are similar to `scanf(3C)`. Their effect is as though `mvwgetstr(3XCURSES)` were called to get a multi-byte character string from the current or specified window at the current or specified cursor position, and then `sscanf()` were used to interpret and convert that string.

### Return Values

Upon successful completion, these functions return `OK`. Otherwise, they return `ERR`.

### Errors

No errors are defined.

### See Also

`getnstr(3XCURSES), printw(3XCURSES), scanf(3C), wcstombs(3C)`
mvwin(3XCURSES)

<table>
<thead>
<tr>
<th>NAME</th>
<th>mvwin – move window</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>#include &lt;curses.h&gt;</td>
</tr>
</tbody>
</table>

```c
int mvwin(WINDOW *win, int y, int x);
```

<table>
<thead>
<tr>
<th>PARAMETERS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>win</code></td>
<td>Is a pointer to the window to move.</td>
</tr>
<tr>
<td><code>y</code></td>
<td>Is the y (row) coordinate of the upper left corner of the window.</td>
</tr>
<tr>
<td><code>x</code></td>
<td>Is the x (column) coordinate of the upper left corner of the window.</td>
</tr>
</tbody>
</table>

| DESCRIPTION | The `mvwin()` function moves the specified window (or subwindow), placing its upper left corner at the positions specified by `x` and `y`. The entire window must fit within the physical boundaries of the screen or an error results. In the case of a subwindow, the window must remain within the boundaries of the parent window. |

<table>
<thead>
<tr>
<th>RETURN VALUES</th>
<th>On success, the <code>mvwin()</code> function returns OK. Otherwise, it returns ERR.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERRORS</td>
<td>None.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>derwin(3XCURSES)</td>
</tr>
</tbody>
</table>
### NAME
napms – sleep process for a specified length of time

### SYNOPSIS
```
#include <curses.h>

int napms(int ms);
```

### PARAMETERS
- **ms**
  
  Is the number of milliseconds to sleep.

### DESCRIPTION
The `napms()` function sleeps for at least `ms` milliseconds.

### RETURN VALUES
The `napms()` function always returns OK.

### ERRORS
None.

### SEE ALSO
`delay_output(3XCURSES)`
newpad, pnoutrefresh, presfresh, subpad – create or refresh a pad or subpad

SYNOPSIS

#include <curses.h>

WINDOW *newpad(int nlines, int ncols);

int pnoutrefresh(WINDOW *pad, int pminrow, int pmincol, int sminrow,
                 int smincol, int smaxrow, int smaxcol);

int presfresh(WINDOW *pad, int pminrow, int pmincol, int sminrow,
              int smincol, int smaxrow, int smaxcol);

WINDOW *subpad(WINDOW *orig, int nlines, int ncols);

PARAMETERS

nlines  Is the number of lines in the pad to be created.
ncols  Is the number of columns in the pad to be created.
    pad  Is a pointer to the pad to refresh.
pminrow  Is the row coordinate of the upper left corner of the pad
         rectangle to be copied
pmincol  Is the column coordinate of the upper left corner of the pad
         rectangle to be copied.
sminrow  Is the row coordinate of the upper left corner of the
         rectangle on the physical screen where pad is to be positioned.
smincol  Is the column coordinate of the upper left corner of the
         rectangle on the physical screen where pad is to be positioned.
smaxrow  Is the row coordinate of the lower right corner of the
         rectangle on the physical screen where the pad is to be positioned.
smaxcol  Is the column coordinate of the lower right corner of the
         rectangle on the physical screen where the pad is to be positioned.
    orig  Is a pointer to the parent pad within which a sub-pad is created.

DESCRIPTION

The newpad() function creates a new pad with the specified number of lines and columns. A pointer to the new pad structure is returned. A pad differs from a window in that it is not restricted to the size of the physical screen. It is useful when only part of a large window will be displayed at any one time.

Automatic refreshes by scrolling or echoing of input do not take place when pads are used. Pads have their own refresh commands, presfresh() and pnoutrefresh().

The presfresh() function copies the specified portion of the logical pad to the terminal screen. The parameters pmincol and pminrow specify the upper left corner of
newpad(3XCURSES)

the rectangular area of the pad to be displayed. The lower right coordinate of the
rectangular area of the pad that is to be displayed is calculated from the screen
parameters (sminrow, smincol, smaxrow, smaxcol).

This function calls the pnoutrefresh() function to copy the specified portion of pad
to the terminal screen and the doupdate(3XCURSES) function to do the actual
update. The logical cursor is copied to the same location in the physical window
unless leaveok(3XCURSES) is enabled (in which case, the cursor is placed in a
position that the program finds convenient).

When outputting several pads at once, it is often more efficient to call the
pnoutrefresh() and doupdate() functions directly. A call to pnoutrefresh() for each pad first, followed by only one call to doupdate() to update the screen,
results in one burst of output, fewer characters sent, and less CPU time used.

The subpad() function creates a sub-pad within the pad orig with the specified
number of lines and columns. A pointer to the new pad structure is returned. The
sub-pad is positioned in the middle of orig. Any changes made to one pad affect the
other. touchwin(3XCURSES) or touchline(3XCURSES) will likely have to be called
on pad orig to correctly update the window.

RETURN VALUES

On success, the newpad() and subpad() functions returns a pointer to the new pad
data structure. Otherwise, they return a null pointer.

On success, the pnoutrefresh() and refresh() functions return OK. Otherwise, they return ERR.

SEE ALSO clearok(3XCURSES), doupdate(3XCURSES), is_linetouched(3XCURSES),
pechochar(3XCURSES)
The `nl()` function enables the handling of newlines. The `nl()` function converts newline into carriage return and line feed on output and converts carriage return into newline on input. `nonl()` disables the handling of newlines.

The handling of newlines is initially enabled. Disabling the handling of newlines results in faster cursor motion since X/Open Curses can use the line-feed capability more efficiently.

On success, these functions return `OK`. Otherwise, they return `ERR`.

None.
nodelay(3XCURSES)

NAME
nodelay – set blocking or non-blocking read

SYNOPSIS
#include <curses.h>
int nodelay(WINDOW *win, bool bf);

PARAMETERS
win Is a pointer to the window in which to enable non-blocking.

bf Is a Boolean expression.

DESCRIPTION
If enabled, (bf is TRUE), the nodelay() function causes getch(3XCURSES) to return ERR if no input is ready. When disabled, getch() blocks until a key is pressed.

RETURN VALUES
On success, the nodelay() function returns OK. Otherwise, it returns ERR.

ERRORS
None.

SEE ALSO
getch(3XCURSES), halfdelay(3XCURSES), notimeout(3XCURSES)
noqiflush(3XCURSES)

NAME
noqiflush, qiflush – control flush of input and output on interrupt

SYNOPSIS
#include <curses.h>

void noqiflush(void);
void qiflush(void);

DESCRIPTION
The qiflush() function enables the flushing of input and output queues when an interrupt, quit, or suspend character is sent to the terminal. The noqiflush() function disables this flushing.

RETURN VALUES
These functions do not return a value.

ERRORS
None

SEE ALSO
flushinp(3XCURSES), intrflush(3XCURSES)
### notimeout(3XCURSES)

**NAME**
notimeout, timeout, wtimeout – set timed blocking or non-blocking read

**SYNOPSIS**
```
#include <curses.h>

int notimeout(WINDOW *win, bool bf);
void timeout(int delay);
void wtimeout(WINDOW win, int delay);
```

**PARAMETERS**
- **win** Is a pointer to the window in which to set the timed blocking.
- **bf** Is a Boolean expression.
- **delay** Is the number of milliseconds to block or wait for input.

**DESCRIPTION**
If `bool` is TRUE, the `notimeout()` function disables a timer used by `getch(3XCURSES)` when handling multibyte function key sequences.

When `bool` is FALSE and keypad handling is enabled, a timer is set by `getch()` to handle bytes received that could be the beginning of a function key (for example, ESC). If the remainder of the sequence is not received before the time expires, the first byte is returned; otherwise, the value of the function key is returned. Subsequent calls to the `getch()` function will return the other bytes received for the incomplete key sequence.

The `timeout()` and `wtimeout()` functions set the length of time `getch()` waits for input for windows `stdscr` and `win`, respectively. These functions are similar to `nodelay(3XCURSES)` except the time to block or wait for input can be specified.

A negative `delay` causes the program to wait indefinitely for input; a `delay` of 0 returns `ERR` if no input is ready; and a positive `delay` blocks until input arrives or the time specified expires, (in which case, `ERR` is returned).

**RETURN VALUES**
On success, the `notimeout()` function returns `OK`. Otherwise, it returns `ERR`.

The `timeout()` and `wtimeout()` functions do not return a value.

**ERRORS**
None.

**SEE ALSO**
`getch(3XCURSES), halfdelay(3XCURSES), nodelay(3XCURSES)`
NAME  overlay, overwrite – copy overlapped windows

SYNOPSIS
#include <curses.h>

int overlay(const WINDOW *srcwin, WINDOW *dstwin);
int overwrite(const WINDOW *srcwin, WINDOW *dstwin);

PARAMETERS
srcwin  Is a pointer to the source window to be copied.
dstwin  Is a pointer to the destination window to be overlayed or overwritten.

DESCRIPTION
The overwrite() and overlay() functions overlay srcwin on top of dstwin. The srcwin and dstwin arguments do not have to be the same size; only text where the two windows overlap is copied.

The overwrite() function copies characters as though a sequence of win_wch(3XCURSES) and wadd_wch(3XCURSES) were performed with the destination window’s attributes and background attributes cleared.

The overlay() function does the same thing, except that, whenever a character to be copied is the background character of the source window, overlay() does not copy the character but merely moves the destination cursor the width of the source background character.

If any portion of the overlaying window border is not the first column of a multi-column character, then all the column positions will be replaced with the background character and rendition before the overlay is done. If the default background character is a multi-column character when this occurs, then these functions fail.

RETURN VALUES
Upon successful completion, these functions return OK. Otherwise, they return ERR.

ERRORS
No errors are defined.

EXAMPLES
EXAMPLE 1 Implement a pop-up dialog

The following example demonstrates the use of overwrite() to implement a pop-up dialog box.

#include <curses.h>
/*
 * Pop-up a window on top of curscr. If row and/or col
 * are -1 then that dimension will be centered within
 * curscr. Return 0 for success or -1 if malloc() failed.
 * Pass back the working window and the saved window for the
 * pop-up. The saved window should not be modified.
 */
int popup(WINDOW **work, **save, nrows, ncols, row, col)
{...}
EXAMPLE 1 Implement a pop-up dialog  
(Continued)

```c
int mr, mc;
getmaxyx(curscr, mr, mc);
/* Windows are limited to the size of curscr. */
if (mr < nrows)
    nrows = mr;
if (mc < ncols)
    ncols = mc;
/* Center dimensions. */
if (row == -1)
    row = (mr-nrows)/2;
if (col == -1)
    col = (mc-ncols)/2;
/* The window must fit entirely in curscr. */
if (mr < row+nrows)
    row = 0;
if (mc < col+ncols)
    col = 0;
*work = newwin(nrows, ncols, row, col);
if (*work == NULL)
    return (-1);
if ((*save = dupwin(*work)) == NULL) {
    delwin(*work);
    return (-1);
}
overwrite(curscr, *save);
return (0);
}

/* Restore the region covered by a pop-up window. */
/* Delete the working window and the saved window. */
/* This function is the complement to popup( ). Return */
/* 0 for success or -1 for an error. */
int
popdown(work, save)
WINDOW *work, *save;
{
    (void) wnoutrefresh(save);
    (void) delwin(save);
    (void) delwin(work);
    return (0);
}

/* Compute the size of a dialog box that would fit around */
/* the string. */
void
dialsize(str, nrows, ncols)
char *str;
int *nrows, *ncols;
{
    int rows, cols, col;
    for (rows = 1, cols = col = 0; *str != '\0'; ++str) {
        if (*str == '\n') {
            /* A newline. */
            cols = col = 0;
        } else {
            /* Add the character to the current line. */
            rows = (rows == 0) ? 1 : rows;
            cols = cols + 1;
        }
    }
    *nrows = (rows > 0) ? rows : 0;
    *ncols = (cols > 0) ? cols : 0;
}
```

Curses Library Functions  311
EXAMPLE 1 Implement a pop-up dialog (Continued)

```c
if (cols < col)
    cols = col;
    col = 0;
++rows;
} else {
    ++col;
}

if (cols < col)
    cols = col;
*nrows = rows;
*ncols = cols;

/* Write a string into a dialog box. */
void
dialfill(w, s)
WINDOW *w;
char *s;
{
    int row;
    (void) wmove(w, 1, 1);
    for (row = 1; *s != '\0'; ++s) {
        (void) waddch(w, *({unsigned char*) s});
        if (*s == '\n')
            wmove(w, ++row, 1);
    }
    box(w, 0, 0);
}

void
dialog(str)
char *str;
{
    WINDOW *work, *save;
    int nrows, ncols, row, col;
    /* Figure out size of window. */
    dialsize(str, &nrows, &ncols);
    /* Create a centered working window with extra */
    /* room for a border. */
    (void) popup(&work, &save, nrows+2, ncols+2, -1, -1);
    /* Write text into the working window. */
    dialfill(work, str);
    /* Pause. Remember that wgetch() will do a wrefresh() */
    /* for us. */
    (void) wgetch(work);
    /* Restore curscr and free windows. */
    (void) popdown(work, save);
    /* Redraw curscr to remove window from physical screen. */
    (void) doupdate( );
}
```

SEE ALSO

`copywin(3XCURSES), wadd_wch(3XCURSES), win_wch(3XCURSES)`

312  man pages section 3: Curses Library Functions • Last Revised 5 May 1999
panel_above(3CURSES)

NAME
panel_above, panel_below – panels deck traversal primitives

SYNOPSIS
c [ flag ... ] file ... -lpanel -lcurses [ library ... ]
#include <panel.h>

PANEL *
panel_above(PANEL *panel);

PANEL *
panel_below(PANEL *panel);

DESCRIPTION
panel_above() returns a pointer to the panel just above panel, or NULL if panel is
the top panel. panel_below() returns a pointer to the panel just below panel, or
NULL if panel is the bottom panel.

If NULL is passed for panel, panel_above() returns a pointer to the bottom panel in
the deck, and panel_below() returns a pointer to the top panel in the deck.

RETURN VALUES
NULL is returned if an error occurs.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), panels(3CURSES), attributes(5)

NOTES
These routines allow traversal of the deck of currently visible panels.

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

Curses Library Functions 313
### NAME

panel_move, move_panel – move a panels window on the virtual screen

### SYNOPSIS

```c
#include <panel.h>

int move_panel(PANEL *panel, int starty, int startx);
```

### DESCRIPTION

`move_panel()` moves the `curses` window associated with `panel` so that its upper left-hand corner is at `starty`, `startx`. See usage note, below.

### RETURN VALUES

OK is returned if the routine completes successfully, otherwise ERR is returned.

### ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

### SEE ALSO

curses(3CURSES), panel_update(3CURSES), panels(3CURSES), attributes(5)

### NOTES

For `panels` windows, use `move_panel()` instead of the `mvwin()` `curses` routine. Otherwise, `update_panels()` will not properly update the virtual screen.

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

---

314 man pages section 3: Curses Library Functions • Last Revised 31 Dec 1996
panel_new(3CURSES)

NAME
panel_new, new_panel, del_panel – create and destroy panels

SYNOPSIS
cce [ flag ... ] file ... -lpanel -lcurses [ library ... ]
#include <panel.h>

PANEL *new_panel(WINDOW *win);
int del_panel(PANEL *panel);

DESCRIPTION
new_panel() creates a new panel associated with win and returns the panel pointer.
The new panel is placed on top of the panel deck.

del_panel() destroys panel, but not its associated window.

RETURN VALUES
new_panel() returns NULL if an error occurs.

del_win() returns OK if successful, ERR otherwise.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), panel_update(3CURSES), panels(3CURSES), attributes(5)

NOTES
The header <panel.h> automatically includes the header <curses.h>.
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 compiled with -lpanel and -lcurses on the cc command line.

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.

The following table lists each panels routine and the name of the manual page on which it is described.

<table>
<thead>
<tr>
<th>panels Routine Name</th>
<th>Manual Page Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>bottom_panel</td>
<td>panel_top(3CURSES)</td>
</tr>
<tr>
<td>del_panel</td>
<td>panel_new(3CURSES)</td>
</tr>
<tr>
<td>hide_panel</td>
<td>panel_show(3CURSES)</td>
</tr>
<tr>
<td>move_panel</td>
<td>panel_move(3CURSES)</td>
</tr>
<tr>
<td>new_panel</td>
<td>panel_new(3CURSES)</td>
</tr>
<tr>
<td>panel_above</td>
<td>panel_above(3CURSES)</td>
</tr>
<tr>
<td>panel_below</td>
<td>panel_above(3CURSES)</td>
</tr>
<tr>
<td>panel_hidden</td>
<td>panel_show(3CURSES)</td>
</tr>
<tr>
<td>panel_userptr</td>
<td>panel_userptr(3CURSES)</td>
</tr>
<tr>
<td>panel_window</td>
<td>panel_window(3CURSES)</td>
</tr>
<tr>
<td>replace_panel</td>
<td>panel_window(3CURSES)</td>
</tr>
<tr>
<td>set_panel_userptr</td>
<td>panel_userptr(3CURSES)</td>
</tr>
<tr>
<td>show_panel</td>
<td>panel_show(3CURSES)</td>
</tr>
<tr>
<td>top_panel</td>
<td>panel_top(3CURSES)</td>
</tr>
<tr>
<td>update_panels</td>
<td>panel_update(3CURSES)</td>
</tr>
</tbody>
</table>
Each `panels` 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.

**ATTRIBUTES**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
curses(3CURSES), attributes(5) and 3X pages whose names begin “panel_” for detailed routine descriptions.

**NOTES**
The header `<panel.h>` automatically includes the header `<curses.h>`.
NAME
panel_show, show_panel, hide_panel, panel_hidden – panels deck manipulation routines

SYNOPSIS
cc [ flag ... ] file ... -lpanel -lcurses [ library ... ]
#include <panel.h>

int show_panel(PANEL *panel);
int hide_panel(PANEL *panel);
int panel_hidden(PANEL *panel);

DESCRIPTION
show_panel() makes panel, previously hidden, visible and places it on top of the
deck of panels.

hide_panel() removes panel from the panel deck and, thus, hides it from view. The
internal data structure of the panel is retained.

panel_hidden() returns TRUE (1) or FALSE (0) indicating whether or not panel is
in the deck of panels.

RETURN VALUES
show_panel() and hide_panel() return the integer OK upon successful
completion or ERR upon error.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
curses(3CURSES), panel_update(3CURSES), panels(3CURSES), attributes(5)

NOTES
The header <panel.h> automatically includes the header <curses.h>.
NAME  panel_top, top_panel, bottom_panel – panels deck manipulation routines

SYNOPSIS

cc [ flag ... ] file ... -lpanel -lcurses [ library ... ]
#include <panel.h>

int top_panel(PANEL *panel);
int bottom_panel(PANEL *panel);

DESCRIPTION  top_panel() pulls panel to the top of the desk of panels. It leaves the size, location, and contents of its associated window unchanged.

bottom_panel() puts panel at the bottom of the deck of panels. It leaves the size, location, and contents of its associated window unchanged.

RETURN VALUES  All of these routines return the integer OK upon successful completion or ERR upon error.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3CURSES), panel_update(3CURSES), panels(3CURSES), attributes(5)

NOTES  The header <panel.h> automatically includes the header <curses.h>.
NAME  panel_update, update_panels – panels virtual screen refresh routine

SYNOPSIS  cc [ flag ... ] file ... -lpanel -lcurses [ library ... ]
            #include <panel.h>
            void update_panels(void);

DESCRIPTION  update_panels() refreshes the virtual screen to reflect the depth relationships
              between the panels in the deck. The user must use the curses library call
doupdate() (see curs_refresh(3CURSES)) to refresh the physical screen.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curs_refresh(3CURSES), curses(3CURSES), panels(3CURSES), attributes(5)

NOTES  The header <panel.h> automatically includes the header <curses.h>.
panel_userptr(3CURSES)

NAME  panel_userptr, set_panel_userptr — associate application data with a panels panel

SYNOPSIS  
```
cc [ flag ... ] file ... -lpanel -lcurses [ library ... ]
#include <panel.h>

int set_panel_userptr(PANEL *panel, char *ptr);
char *panel_userptr(PANEL *panel);
```

DESCRIPTION  Each panel has a user pointer available for maintaining relevant information.

set_panel_userptr() sets the user pointer of panel to ptr.

panel_userptr() returns the user pointer of panel.

RETURN VALUES  set_panel_userptr returns OK if successful, ERR otherwise.

panel_userptr returns NULL if there is no user pointer assigned to panel.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3CURSES), panels(3CURSES), attributes(5)

NOTES  The header <panel.h> automatically includes the header <curses.h>.
panel_window(3CURSES)

NAME  panel_window, replace_panel – get or set the current window of a panels panel

SYNOPSIS  cc [ flag ... ] file ... -lpanel -lcurses [ library ... ]
#include <panel.h>

WINDOW *panel_window(PANEL *panel);
int replace_panel(PANEL *panel, WINDOW *win);

DESCRIPTION  panel_window() returns a pointer to the window of panel.

replace_panel() replaces the current window of panel with win.

RETURN VALUES  panel_window() returns NULL on failure.

replace_panel() returns OK on successful completion, ERR otherwise.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  curses(3CURSES), panels(3CURSES), attributes(5)

NOTES  The header <panel.h> automatically includes the header <curses.h>.
pechochar, pecho_wchar – add character and refresh window

SYNOPSIS

```c
#include <curses.h>

int pechochar(WINDOW *pad, chtype ch);
int pecho_wchar(WINDOW *pad, const chtype *wch);
```

PARAMETERS

- `pad`: Is a pointer to the pad in which the character is to be added.
- `ch`: Is a pointer to the character to be written to the pad.
- `wch`: Is a pointer to the complex character to be written to the pad.

DESCRIPTION

The `pechochar()` function is equivalent to calling `waddch(3XCURSES)` followed by a call to `prefresh(3XCURSES)`. The `pecho_wchar()` function is equivalent to calling `wadd_wch(3XCURSES)` followed by a call to `prefresh()`. `prefresh()` reuses the last position of the pad on the screen for its parameters.

RETURN VALUES

On success, these functions return `OK`. Otherwise, they return `ERR`.

ERRORS

None.

SEE ALSO

`add_wch(3XCURSES)`, `addch(3XCURSES)`, `newpad(3XCURSES)`
These functions generate graphics output for a set of output devices. The format of the output is dependent upon which link editor option is used when the program is compiled and linked (see Link Editor).

The term "current point" refers to the current setting for the x and y coordinates.

The arc() function specifies a circular arc. The coordinates \((x_0, y_0)\) specify the center of the arc. The coordinates \((x_1, y_1)\) specify the starting point of the arc. The coordinates \((x_2, y_2)\) specify the end point of the circular arc.

The box() function specifies a rectangle with coordinates \((x_0, y_0), (x_0, y_1), (x_1, y_0), (x_1, y_1)\). The current point is set to \((x_1, y_1)\).

The circle() function specifies a circle with a center at the coordinates \((x, y)\) and a radius of \(r\).

The closevt() and closepl() functions flush the output.
The `cont()` function specifies a line beginning at the current point and ending at the coordinates \((x, y)\). The current point is set to \((x, y)\).

The `erase()` function starts another frame of output.

The `label()` function places the null terminated string `s` so that the first character falls on the current point. The string is then terminated by a NEWLINE character.

The `line()` function draws a line starting at the coordinates \((x0, y0)\) and ending at the coordinates \((x1, y1)\). The current point is set to \((x1, y1)\).

The `linmod()` function specifies the style for drawing future lines. `s` may contain one of the following: `dotted`, `solid`, `longdashed`, `shortdashed`, or `dotdashed`.

The `move()` function sets the current point to the coordinates \((x, y)\).

The `openpl()` or `openvt()` function must be called to open the device before any other `plot` functions are called.

The `point()` function plots the point given by the coordinates \((x, y)\). The current point is set to \((x, y)\).

The `space()` function specifies the size of the plotting area. The plot will be reduced or enlarged as necessary to fit the area specified. The coordinates \((x0, y0)\) specify the lower left hand corner of the plotting area. The coordinates \((x1, y1)\) specify the upper right hand corner of the plotting area.

**Link Editor**

Various flavors of these functions exist for different output devices. They are obtained by using the following `ld(1)` options:

- `-lplot` device-independent graphics stream on standard output in the format described in `plot(4B)`
- `-l300` GSI 300 terminal
- `-l300s` GSI 300S terminal
- `-l4014` Tektronix 4014 terminal
- `-l450` GSI 450 terminal
- `-lvt0`

**FILES**

- `/usr/lib/libplot.a` archive library
- `/usr/lib/libplot.so.1` shared object
- `/usr/lib/sparcv9/libplot.so.1` 64-bit shared object
- `/usr/lib/lib300.a` archive library
plot(3PLOT)

/usr/lib/lib300.so.1
  shared object
/usr/lib/sparcv9/lib300.so.1
  64-bit shared object
/usr/lib/lib300s.a
  archive library
/usr/lib/lib300s.so.1
  shared object
/usr/lib/sparcv9/lib300s.so.1
  64-bit shared object
/usr/lib/lib4014.a
  archive library
/usr/lib/lib4014.so.1
  shared object
/usr/lib/sparcv9/lib4014.so.1
  64-bit shared object
/usr/lib/lib450.a
  archive library
/usr/lib/lib450.so.1
  shared object
/usr/lib/sparcv9/lib450.so.1
  64-bit shared object
/usr/lib/libvt0.a
  archive library
/usr/lib/libvt0.so.1
  shared object
/usr/lib/sparcv9/libvt0.so.1
  64-bit shared object

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

graph(1), ld(1), libplot(3LIB), plot(4B), attributes(5)
NAME
putp, tputs – apply padding information and output string

SYNOPSIS
#include <curses.h>

int putp(const char *str);
int tputs(const char *str, int affcnt, int (*putfunc)(int));

PARAMETERS
str Is a pointer to a terminfo variable or return value
from tgetstr(3XCURSES), tgoto(3XCURSES),
tigetstr(3XCURSES), or tparm(3XCURSES).
affcnt Is the number of lines affected, or 1 if not relevant.
putfunc Is the output function.

DESCRIPTION
The putp() and tputs() functions are low-level functions used to deal directly with
the terminfo database. The use of appropriate X/Open Curses functions is
recommended for most situations.

The tputs() function adds padding information and then outputs str. str must be a
terminfo string variable or the result value from tgetstr(), tgoto(),
tigetstr(), or tparm(). The tputs() function replaces the padding specification
(if one exists) with enough characters to produce the specified delay. Characters are
output one at a time to putfunc, a user-specified function similar to putchar(3C).

The putp() function calls tputs() as follows:
tputs(str, 1, putchar)

RETURN VALUES
On success, these functions return OK.

ERRORS
None.

USAGE
The output of putp() goes to stdout, not to the file descriptor, fildes, specified in
setupterm(3XCURSES).

SEE ALSO
putchar(3C), setupterm(3XCURSES), tgetent(3XCURSES),
tigetflag(3XCURSES), terminfo(4)
redrawwin(3XCURSES)

<table>
<thead>
<tr>
<th>NAME</th>
<th>redrawwin, wredrawln – redraw screen or portion of screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>#include &lt;curses.h&gt;</td>
</tr>
<tr>
<td></td>
<td>int redrawwin(WINDOW *win);</td>
</tr>
<tr>
<td></td>
<td>int wredrawln(WINDOW *win, int beg_line, int num_lines);</td>
</tr>
<tr>
<td>PARAMETERS</td>
<td>win Is a pointer to the window in which to redraw.</td>
</tr>
<tr>
<td></td>
<td>beg_line Is the first line to redraw.</td>
</tr>
<tr>
<td></td>
<td>num_lines Is the number of lines to redraw.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The redrawwin() and wredrawln() functions force portions of a window to be redrawn to the terminal when the next refresh operation is performed.</td>
</tr>
<tr>
<td></td>
<td>The redrawwin() function forces the entire window win to be redrawn, while the wredrawln() function forces only num_lines lines starting with beg_line to be redrawn. Normally, refresh operations use optimization methods to reduce the actual amount of the screen to redraw based on the current screen contents. These functions tell the refresh operations not to attempt any optimization when redrawing the indicated areas.</td>
</tr>
<tr>
<td></td>
<td>These functions are useful when the data that exists on the screen is believed to be corrupt and for applications such as screen editors that redraw portions of the screen.</td>
</tr>
<tr>
<td>RETURN VALUES</td>
<td>On success, these functions return OK. Otherwise, they return ERR.</td>
</tr>
<tr>
<td>ERRORS</td>
<td>None.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>doupdate(3XCURSES)</td>
</tr>
</tbody>
</table>
### NAME
resetty, savetty – restore/save terminal modes

### SYNOPSIS
```
#include <curses.h>

int resetty(void);

int savetty(void);
```

### DESCRIPTION
The savetty() and resetty() functions save and restore the terminal state, respectively. The savetty() function saves the current state in a buffer; the resetty() function restores the state to that stored in the buffer at the time of the last savetty() call.

### RETURN VALUES
On success, these functions return OK. Otherwise, they return ERR.

### ERRORS
None.
NAME    ripoffline – reserve screen line for dedicated purpose

SYNOPSIS  

```c
#include <curses.h>

int ripoffline(int line, int (*init)(WINDOW *win, int width));
```

PARAMETERS  

- `line`: determines whether the screen line being reserved comes from the top of `stdscr` (line is positive) or the bottom (line is negative).
- `init`: Is a pointer to a function that initializes the one-line window.
- `win`: Is a pointer to one-line window created by this function.
- `width`: Is the number of columns in the window pointed to by the `win` parameter.

DESCRIPTION  

The `ripoffline()` function reserves a screen line as a one line window. To use this function, it must be called before you call `initscr(3XCURSES)` or `newterm(3XCURSES)`. When `initscr()` or `newterm()` is called, so is the function pointed to by `init`. The function pointed to by `init` takes two arguments: a pointer to the one-line window and the number of columns in that window. This function cannot use the LINES or COLS variables and cannot call `wrefresh(3XCURSES)` or `doupdate(3XCURSES)`, but may call `wnoutrefresh(3XCURSES)`.

RETURN VALUES  

The `ripoffline()` function always returns `OK`.

ERRORS  

None.

SEE ALSO  

`doupdate(3XCURSES)`, `initscr(3XCURSES)`, `slk_attroff(3XCURSES)`
### NAME

scr_dump, scr_init, scr_restore, scr_set – write screen contents to/from a file

### SYNOPSIS

```c
#include <curses.h>

int scr_dump(const char *filename);
int scr_init(const char *filename);
int scr_restore(const char *filename);
int scr_set(const char *filename);
```

### PARAMETERS

#### filename

Is a pointer to the file in which screen contents are written.

### DESCRIPTION

These function perform input/output functions on a screen basis.

The `scr_dump()` function writes the contents of the virtual screen, `curscr`, to `filename`.

The `scr_restore()` function reads the contents of `filename` from `curscr` (which must have been written with `scr_dump()`). The next refresh operation restores the screen to the way it looks in `filename`.

The `scr_init()` function reads the contents of `filename` and uses those contents to initialize the X/Open Curses data structures to what is actually on screen. The next refresh operation bases its updates on this data, unless the terminal has been written to since `filename` was saved or the `terminfo` capabilities `rmcup` and `nrrmc` are defined for the current terminal.

The `scr_set()` function combines `scr_restore()` and `scr_init()`. It informs the program that the contents of the file `filename` are what is currently on the screen and that the program wants those contents on the screen.

### RETURN VALUES

On success, these functions return `OK`. Otherwise, they return `ERR`.

### ERRORS

None.

### SEE ALSO

delscreen(3XCURSES), douupdate(3XCURSES), endwin(3XCURSES), getwin(3XCURSES)
# scrl(3XCURSES)

## NAME
scrl, scroll, wscrl – scroll a window

## SYNOPSIS
```
#include <curses.h>

int scrl(int n);
int scroll(WINDOW *win);
int wscrl(WINDOW *win, int n);
```

## PARAMETERS
- **n**: number and direction of lines to scroll
- **win**: pointer to the window in which to scroll

## DESCRIPTION
The `scroll()` function scrolls the window `win` up one line. The current cursor position is not changed.

The `scrl()` and `wscrl()` functions scroll the window `stdscr` or `win` up or down `n` lines, where `n` is a positive (scroll up) or negative (scroll down) integer.

The `scrollok(3XCURSES)` function must be enabled for these functions to work.

## RETURN VALUES
- On success, these functions return `OK`. Otherwise, they return `ERR`.

## ERRORS
None.

## SEE ALSO
- clearok(3XCURSES)
NAME  setcchar – set a cchar_t type character from a wide character and rendition

SYNOPSIS  

#include <curses.h>

int setcchar(cchar_t *wcval, const wchar_t *wch, const attr_t attrs, 
short color_pair, const void *opts);

PARAMETERS  

wcval        Is a pointer to a location where a cchar_t character 
(and its rendition) can be stored.

wch          Is a pointer to a wide character.

attrs        Is the set of attributes to apply to wch in creating wcval.

color_pair   Is the color pair to apply to wch in creating wcval.

opts         Is reserved for future use. Currently, this must be a null pointer.

DESCRIPTION  The setcchar() function takes the wide character pointed to by wch, combines it 
with the attributes indicated by attrs and the color pair indicated by color_pair and 
stores the result in the object pointed to by wcval.

RETURN VALUES  On success, the setcchar() function returns OK. Otherwise, it returns ERR.

ERRORS        None.

SEE ALSO      attroff(3XCURSES), can_change_color(3XCURSES), getcchar(3XCURSES)
set_term(3XCURSES)

#NAME
set_term – switch between terminals

#SYNOPSIS
#include <curses.h>

SCREEN *set_term(SCREEN *new);

#PARAMETERS
new
Is the new terminal to which the set_term() function will switch.

#DESCRIPTION
The set_term() function switches to the terminal specified by new and returns a screen reference to the previous terminal. Calls to subsequent X/Open Curses functions affect the new terminal.

#RETURN VALUES
On success, the set_term() function returns a pointer to the previous screen. Otherwise, it returns a null pointer.

#ERRORS
None.
NAME

slk_attroff, slk_attr_off, slk_attron, slk_attr_on, slk_attrset, slk_attr_set, slk_clear,
slk_color, slk_init, slk_label, slk_noutrefresh, slk_refresh, slk_restore, slk_set,
slk_touch, slk_wset – soft label functions

SYNOPSIS

#include <curses.h>

int slk_attroff(const chtype attrs);
int slk_attr_off(const attr_t attrs, void *opts);
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_number, void *opts);
int slk_clear(void);
int slk_color(short color_pair_number);
int slk_init(int fmt);
char *slk_label(int labnum);
int slk_noutrefresh(void);
int slk_refresh(void);
int slk_restore(void);
int slk_set(int labnum, const char *label, int justify);
int slk_touch(void);
int slk_wset(int labnum, const wchar_t *label, int justify);

PARAMETERS

attrs are the window attributes to be added or removed.

opts Is reserved for future use. Currently, this must be a null pointer.

color_pair_number Is a color pair.

fmt Is the format of how the labels are arranged on the screen.

labnum Is the number of the soft label.

label Is the name to be given to a soft label.

justify Is a number indicating how to justify the label name.

DESCRIPTION

The Curses interface manipulates the set of soft function-key labels that exist on many terminals. For those terminals that do not have soft labels, Curses takes over the bottom line of stdscr, reducing the size of stdscr and the value of the LINES external variable. There can be up to eight labels of up to eight display columns each.
To use soft labels, `slk_init()` must be called before calling `initscr(3XCURSES)`, `newterm(3XCURSES)`, or `ripoffline(3XCURSES)`. If `initscr()` eventually uses a line from `stdscr` to emulate the soft labels, then `fmt` determines how the labels are arranged on the screen. Setting `fmt` to 0 indicates a 3-2-3 arrangement of the labels; 1 indicates a 4-4 arrangement. Other values for `fmt` are unspecified.

The `slk_init()` function has the effect of calling `ripoffline()` to reserve one screen line to accommodate the requested format.

The `slk_set()` and `slk_wset()` functions specify the text of soft label number `labnum`, within the range from 1 to and including 8. The `label` argument is the string to be put the label. With `slk_set()` and `slk_wset()`, the width of the label is limited to eight columns positions. A null string or a null pointer specifies a blank label. The `justify` argument can have the following values to indicate how to justify `label` within the space reserved for it:

0   Align the start of `label` with the start of the space
1   Center `label` within the space
2   Align the end of `label` with the end of the space

The `slk_refresh()` and `slk_noutrefresh()` functions correspond to the `wrefresh(3XCURSES)` and `wnoutrefresh(3XCURSES)` functions.

The `slk_label()` function obtains soft label number `labnum`.

The `slk_clear()` function immediately clears the soft labels from the screen.

The `slk_restore()` function immediately restores the soft labels to the screen after a call to `slk_clear()`.

The `slk_touch()` function forces all the soft labels to be output the next time `slk_refresh()` or `slk_noutrefresh()` is called.

The `slk_attron()`, `slk_attrset()`, and `slk_attroff()` functions correspond to the `attron(3XCURSES)`, `attrset(3XCURSES)`, and `attroff(3XCURSES)` functions. They have an effect only if soft labels are stimulated on the bottom line of the screen.

The `slk_attr_on()`, `slk_attr_off()`, `slk_attr_set()` and `slk_color()` functions correspond to the `attr_on(3XCURSES)`, `attr_off(3XCURSES)`, `attr_set(3XCURSES)`, and `color_set(3XCURSES)` functions. As a result, they support color and the attribute constants with the `WA_` prefix.

The `opts` argument is reserved for definition in a future release. Currently, the `opts` argument is a null pointer.

**RETURN VALUES**

Upon successful completion, the `slk_label()` function returns the requested label with leading and trailing blanks stripped. Otherwise, it returns a null pointer.
Upon successful completion, the other functions return OK. Otherwise, they return ERR.

**ERRORS**

No errors are defined.

**USAGE**

When using multi-byte character sets, applications should check the width of the string by calling mbstowcs(3C) and then wcswidth(3C) before calling slk_set(). When using wide characters, applications should check the width of the string by calling wcwidth() before calling slk_set().

Since the number of columns that a wide string will occupy is codeset-specific, call wcwidth(3C) and wcswidth(3C) to check the number of column positions in the string before calling slk_wset().

Most applications would use slk_noutrefresh() because a wrefresh() is likely to follow soon.

**SEE ALSO**

attr_get(3XCURSES), attroff(3XCURSES), delscreen(3XCURSES), mbstowcs(3C), ripoffline(3XCURSES), wcswidth(3C), wcwidth(3C)
standend(3XCURSES)

NAME standend, standout, wstandend, wstandout – set/clear window attributes

SYNOPSIS #include <curses.h>

int standend(void);
int standout(void);
int wstandend(WINDOW *win);
int wstandout(WINDOW *win);

PARAMETERS win Is a pointer to the window in which attribute changes are to be made.

DESCRIPTION The standend() and wstandend() functions turn off all attributes associated with stdscr and win respectively.

The standout() and wstandout() functions turn on the A_STANDOUT attribute of stdscr and win respectively.

RETURN VALUES These functions always return 1.

ERRORS None.

SEE ALSO attr_get(3XCURSES), attroff(3XCURSES)
NAME
stdscr – default window

SYNOPSIS
#include <curses.h>

extern WINDOW *stdscr;

DESCRIPTION
The external variable stdscr specifies the default window used by functions that do not specify a window using an argument of type WINDOW *. Other windows may be created using newwin().

SEE ALSO
newwin(3XCURSES)
NAME
syncok, wcursyncup, wsyncdown, wsyncup – synchronize window with its parents or children

SYNOPSIS
#include <curses.h>

int syncok(WINDOW *win, bool bf);
void wcursyncup(WINDOW *win);
void wsyncdown(WINDOW *win);
void wsyncup(WINDOW *win);

PARAMETERS
win Is a pointer to a window.
bf Is a Boolean expression.

DESCRIPTION
The syncok() function uses the value of bf to determine whether or not the window win's ancestors are implicitly touched whenever there is a change to win. If bf is TRUE, this touching occurs. If bf is FALSE, it does not occur. The initial value for bf is FALSE.

The wcursyncup() function moves the cursor in win's ancestors to match its position in win.

The wsyncdown() function touches win if any of its ancestors have been touched.

The wsyncup() function touches all ancestors of win.

RETURN VALUES
On success, the syncok() function returns OK. Otherwise, it returns ERR.

The other functions do not return a value.

ERRORS
None.

SEE ALSO
derwin(3XCURSES), doupdate(3XCURSES), is_linetouched(3XCURSES)
NAME
termattrs, term_attrs – get supported terminal video attributes

SYNOPSIS
#include <curses.h>

ctype termattrs(void);
attr_t term_attrs(void);

DESCRIPTION
The termattrs() function extracts the video attributes of the current terminal which is supported by the ctype data type.

The term_attrs() function extracts information for the video attributes of the current terminal which is supported for a cchar_t.

RETURN VALUES
The termattrs() function returns a logical OR of A_values of all video attributes supported by the terminal.

The term_attrs() function returns a logical OR of WA_values of all video attributes supported by the terminal.

ERRORS
No errors are defined.

SEE ALSO
attr_get(3XCURSES), attroff(3XCURSES)
### NAME
`termname` – return the value of the environmental variable TERM

### SYNOPSIS
```c
#include <curses.h>

char *termname(void);
```

### DESCRIPTION
The `termname()` function returns a pointer to the value of the environmental variable TERM (truncated to 14 characters).

### RETURN VALUES
The `termname()` returns a pointer to the terminal’s name.

### ERRORS
None.

### SEE ALSO
`del_curterm(3XCURSES)`
tgetent(3XCURSES)

NAME

   tgetent, tgetflag, tgetnum, tgetstr, tgoto – emulate the termcap database

SYNOPSIS

   #include <term.h>

   int tgetent(char *bp, const char *name);
   int tgetflag(char id[2]);
   int tgetnum(char id[2]);
   char *tgetstr(char id[2], char **area);
   char *tgoto(char *cap, int col, int row);

PARAMETERS

   bp         Is a pointer to a buffer. This parameter is ignored.
   name       Is the termcap entry to look up.
   cap        Is the pointer to a termcap capability.
   area       Is a pointer to the area where tgetstr() stores the decoded
               string.
   col        Is the column placement of the new cursor.
   row        Is the row placement of the new cursor.

DESCRIPTION

   The tgetent() function looks up the termcap entry for name. The emulation
   ignores the buffer pointer bp.

   The tgetflag() function gets the Boolean entry for id.

   The tgetnum() function gets the numeric entry for id.

   The tgetstr() function gets the string entry for id. If area is not a null pointer
   and does not point to a null pointer, tgetstr() copies the string entry into the buffer
   pointed to by *area and advances the variable pointed to by area to the first byte after
   the copy of the string entry.

   The tgoto() function instantiates the parameters col and row into the capability cap
   and returns a pointer to the resulting string.

   All of the information available in the terminfo database need not be available
   through these functions.

RETURN VALUES

   Upon successful completion, those functions that return integers return OK. Otherwise,
   they return ERR.

   Those functions that return pointers return a null pointer when an error occurs.

ERRORS

   No errors are defined.

USAGE

   These functions are included as a conversion aid for programs that use the termcap
   library. Their arguments are the same and the functions are emulated using the
   terminfo database.
tgetent(3XCURSES)

These functions are only guaranteed to operate reliably on character sets in which each character fits into a single byte, whose attributes can be expressed using only constants with the \_A_ prefix.

Any terminal capabilities from the terminfo database that cannot be retrieved using these functions can be retrieved using the functions described on the tigetflag(3XCURSES) manual page.

Portable applications must use tputs(3XCURSES) to output the strings returned by tgetstr() and tgoto().

SEE ALSO putp(3XCURSES), setupterm(3XCURSES), tigetflag(3XCURSES)
tigetflag(3XCURSES)

NAME
tigetflag, tigetnum, tigetstr, tparm – return the value of a terminfo capability

SYNOPSIS
#include <term.h>

int tigetflag(char *capname);
int tigetnum(char *capname);
char *tigetstr(char *capname);
char *tparm(char *cap, long p1, long p2, long p3, long p4, long p5,
            long p6, long p7, long p8, long p9);

PARAMETERS
    capname        Is the name of the terminfo capability for which the value is required.
    cap             Is a pointer to a string capability.
    p1...p9        Are the parameters to be instantiated.

DESCRIPTION
The tigetflag(), tigetnum(), and tigetstr() functions return values for terminfo capabilities passed to them.

The following null-terminated arrays contain the capnames, the termcap codes and full C names for each of the terminfo variables.

char *boolnames, *boolcodes, *boolfnames
char *numnames, *numcodes, *numfnames
char *strnames, *strcodes, *strfnames

The tparm() function instantiates a parameterized string using nine arguments. The string is suitable for output processing by tputs().

RETURN VALUES
On success, the tigetflag(), tigetnum(), and tigetstr() functions return the specified terminfo capability.

tigetflag() returns −1 if capname is not a Boolean capability.

tigetnum() returns −2 if capname is not a numeric capability.

tigetstr() returns (char *)−1 if capname is not a string capability.

On success, the tparm() function returns cap in a static buffer with the parameterization resolved. Otherwise, it returns a null pointer.

ERRORS
None.

SEE ALSO
tgetent(3XCURSES), terminfo(4)
typeahead(3XCURSES)

NAME | typeahead – check for type-ahead characters

SYNOPSIS | #include <curses.h>
            int typeahead(int fd);

PARAMETERS | fd Is the file descriptor that is used to check for
type-ahead characters.

DESCRIPTION | The typeahead() function specifies the file descriptor (fd) to use to check for
type-ahead characters (characters typed by the user but not yet processed by X/Open Curses).

X/Open Curses checks for type-ahead characters periodically while updating the screen. If characters are found, the current update is postponed until the next refresh(3XCURSES) or doupdate(3XCURSES). This speeds up response to commands that have been typed ahead. Normally, the input file pointer passed to newterm(3XCURSES), or stdin in the case of initscr(3XCURSES), is used for type-ahead checking.

If fd is -1, no type-ahead checking is done.

RETURN VALUES | On success, the typeahead() function returns OK. Otherwise, it returns ERR.

ERRORS | None.

SEE ALSO | doupdate(3XCURSES), getch(3XCURSES), initscr(3XCURSES)
NAME
unctrl – generate printable representation of a character

SYNOPSIS
#include <unctrl.h>

char *unctrl(chtype c);

PARAMETERS
c Is a character.

DESCRIPTION
The unctrl() function generates a character string that is a printable representation of c. If c is a control character, it is converted to the ^X notation. If c contains rendition information, the effect is undefined.

RETURN VALUES
Upon successful completion, the unctrl() function returns the generated string. Otherwise, it returns a null pointer.

ERRORS
No errors are defined.

SEE ALSO
addch(3XCURSES), addstr(3XCURSES), wunctrl(3XCURSES)
ungetch(3XCURSES)

NAME | ungetch, unget_wch — push character back onto the input queue

SYNOPSIS |

```c
#include <curses.h>

int ungetch(int ch);
int unget_wch(const wchar_t wch);
```

PARAMETERS |

- `ch` Is the single byte character to be put back in the input queue for the next call to `getch(3XCURSES)`.
- `wch` Is the wide character to be put back in the input queue for the next call to `get_wch(3XCURSES)`.

DESCRIPTION |

The `ungetch()` function pushes `ch` back onto the input queue until the next call to `getch()`.

The `unget_wch()` function is similar to `ungetch()` except that `ch` can be of type `wchar_t`.

RETURN VALUES |

On success, these functions return `OK`. Otherwise, they return `ERR`.

ERRORS |

None.

SEE ALSO |

`get_wch(3XCURSES), getch(3XCURSES)`
use_env(3XCURSES)

NAME
use_env – specify source of screen size information

SYNOPSIS
#include <curses.h>

void use_env(bool boolval);

PARAMETERS
boolval Is a Boolean expression.

DESCRIPTION
The use_env() function specifies the technique by which the implementation
determines the size of the screen. If boolval is FALSE, the implementation uses the
values of lines and columns specified in the terminfo database. If boolval is TRUE, the
implementation uses the LINES and COLUMNS environmental variables. The initial
value is TRUE.

Any call to use_env() must precede calls to initscr(3XCURSES),
newterm(3XCURSES), or setupterm(3XCURSES).

RETURN VALUES
The use_env() function does not return a value.

ERRORS
No errors are defined.

SEE ALSO
del_curterm(3XCURSES), initscr(3XCURSES)
NAME
vidattr, vid_attr, vidputs, vid_puts – output attributes to the terminal

SYNOPSIS
#include <curses.h>

int vidattr(chtype attr);
int vid_attr(attr_t attr, short color_pair_number, void *opt);
int vidputs(chtype attr, int (*putfunc) (int));
int vid_puts(attr_t attr, short color_pair_number, void *opt, int (*putfunc) (int));

PARAMETERS
attr Is the rendition of the foreground window.
color_pair_number Is a color pair.
opt Is reserved for future use. Currently, this must be a null pointer.
putfunc Is a user-supplied output function.

DESCRIPTION
These functions output commands to the terminal that change the terminal’s attributes.

If the terminfo database indicates that the terminal in use can display characters in
the rendition specified by attr, then vidattr() outputs one or more commands to
request that the terminal display subsequent characters in that rendition. The function
outputs by calling putchar(3C). The vidattr() function neither relies on your
updates the model which Curses maintains of the prior rendition mode.

The vidputs() function computes the terminal output string that vidattr() does,
based on attr, but vidputs() outputs by calling the user-supplied function putfunc.
The vid_attr() and vid_puts() functions correspond to vidattr() and
vidputs() respectively, but take a set of arguments, one of type attr_t for the
attributes, one of type short for the color pair number, and a void *, and thus
support the attribute constants with the WA_ prefix.

The opts argument is reserved for definition in a future release. Currently, it is
implemented as a null pointer.

The user-supplied function putfunc (which can be specified as an argument to either
vidputs() or vid_puts()) is either putchar() or some other function with the
same prototype. Both the vidputs() and vid_puts() functions ignore the return
value of putfunc.

RETURN VALUES
Upon successful completion, these functions return OK. Otherwise, they return ERR.

ERRORS
No errors are defined.

USAGE
After use of any of these functions, the model Curses maintains of the state of the
terminal might not match the actual state of the terminal. The application should
touch and refresh the window before resuming conventional use of Curses.
Of these functions requires that the application contain so much information about a particular class of terminal that it defeats the purpose of using Curses.

On some terminals, a command to change rendition conceptually occupies space in the screen buffer (with or without width). Thus, a command to set the terminal to a new rendition would change the rendition of some characters already displayed.

SEE ALSO doupdate(3XCURSES), is_linetouched(3XCURSES), putchar(3C), tigetflag(3XCURSES)
vw_printw(3XCURSES)

NAME | vw_printw – print formatted output in window

SYNOPSIS
#include <stdarg.h>
#include <curses.h>

int vw_printw(WINDOW *win, char *fmt, va_list varglist);

PARAMETERS
fmt
Is a printf() format string.

varglist
Is a pointer to a list of parameters.

win
Is a pointer to the window in which the string is to be written.

DESCRIPTION
The vw_printw() function achieves the same effect as wprintw(3XCURSES) using a variable argument list. The third argument is a va_list, as defined in <stdarg.h>.

RETURN VALUES
Upon successful completion, vw_printw() returns OK. Otherwise, it returns ERR.

ERRORS
No errors are defined.

USAGE
The vw_printw() function is preferred over vwprintw(3XCURSES). The use of the vwprintw() and vw_printw() in the same file will not work, due to the requirements to include <varargs.h> and <stdarg.h>, which both contain definitions of va_list.

SEE ALSO
mvprintw(3XCURSES), printf(3C)
NAME | vwprintw – print formatted output in window

SYNOPSIS | 
#include <varargs.h>
#include <curses.h>

int vwprintw(WINDOW *win, char *fmt, va_list varglist);

PARAMETERS | 
fmt Is a printf() format string.

carglist Is a pointer to a list of parameters.

win Is a pointer to the window in which the string is to be written.

DESCRIPTION | The vwprintw() function achieves the same effect as wprintw(3XCURSES) using a variable argument list. The third argument is a va_list, as defined in <varargs.h>.

RETURN VALUES | Upon successful completion, vwprintw() returns OK. Otherwise, it returns ERR.

ERRORS | No errors are defined.

USAGE | The vwprintw() function is deprecated; the vw_printw(3XCURSES) function is preferred. The use of the vwprintw() and vw_printw() in the same file will not work, due to the requirements to include <varargs.h> and <stdarg.h>, which both contain definitions of va_list.

SEE ALSO | mvprintw(3XCURSES), printf(3C), vw_printw(3XCURSES)
vw_scanw(3XCURSES)

NAME    vw_scanw – convert formatted input from a window

SYNOPSIS  
```
#include <stdarg.h>
#include <curses.h>

int vw_scanw(WINDOW *win, char *fmt, va_list varglist);
```

PARAMETERS  
- `fmt`: Is a scanf() format string.
- `varglist`: Is a pointer to a list of parameters.
- `win`: Is a pointer to the window in which the character is to be read.

DESCRIPTION  
The `vw_scanw()` function achieves the same effect as wscanw(3XCURSES) using a variable argument list. The third argument is a va_list, as defined in `<stdarg.h>`.  

RETURN VALUES  
Upon successful completion, `vw_scanw()` returns OK. Otherwise, it returns ERR.

ERRORS  
No errors are defined.

USAGE  
The `vw_scanw()` function is preferred over vwscanw(3XCURSES). The use of the vwscanw() and vw_scanw() in the same file will not work, due to the requirements to include `<varargs.h>` and `<stdarg.h>`, which both contain definitions of va_list.

SEE ALSO  
mvscanw(3XCURSES), scanf(3C)
vwscanw - convert formatted input from a window

SYNOPSIS

```c
#include <stdarg.h>
#include <curses.h>

int vw_scanw(WINDOW *win, char *fmt, va_list varglist);
```

PARAMETERS

- `fmt` is a `scanf()` format string.
- `varglist` is a pointer to a list of parameters.
- `win` is a pointer to the window in which the character is to be read.

DESCRIPTION

The `vwscanw()` function achieves the same effect as `wscanw(3XCURSES)` using a variable argument list. The third argument is a `va_list`, as defined in `<varargs.h>`.

RETURN VALUES

Upon successful completion, `vwscanw()` returns `OK`. Otherwise, it returns `ERR`.

ERRORS

No errors are defined.

USAGE

The `vwscanw()` function is deprecated; the `vw_scanw(3XCURSES)` function is preferred. The use of the `vwscanw()` and `vw_scanw()` in the same file will not work, due to the requirements to include `<varargs.h>` and `<stdarg.h>`, which both contain definitions of `va_list`.

SEE ALSO

`mvscanw(3XCURSES), scanf(3C), vw_scanw(3XCURSES)`
wunctrl(3XCURSES)

<table>
<thead>
<tr>
<th>NAME</th>
<th>wunctrl – generate printable representation of a wide character</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>#include &lt;curses.h&gt;</td>
</tr>
<tr>
<td></td>
<td>wchar_t *wunctrl(cchar_t *wc);</td>
</tr>
<tr>
<td>PARAMETERS</td>
<td>wc Is a pointer to the wide character.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The wunctrl() function converts the a wide character string that is a printable representation of the wide character wc.</td>
</tr>
<tr>
<td></td>
<td>This function also performs the following processing on the input argument:</td>
</tr>
<tr>
<td></td>
<td>■ Control characters are converted to the ^X notation</td>
</tr>
<tr>
<td></td>
<td>■ Any rendition information is removed.</td>
</tr>
<tr>
<td>RETURN VALUES</td>
<td>Upon successful completion, the wunctrl() function returns the generated string. Otherwise, it returns a null pointer.</td>
</tr>
<tr>
<td>ERRORS</td>
<td>No errors are defined.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>keyname(3XCURSES), unctrl(3XCURSES)</td>
</tr>
</tbody>
</table>
Index

A
activate audio-visual alarm
— beep, 33
— flash, 33
add a string of wchar_t characters to a curses
window and advance cursor —
curs_addwstr, 64
add a wchar_t character (with attributes) to a
curses window and advance cursor —
curs_addwch, 59
add string of wchar_t characters (and attributes)
to a curses window — curs_addwchstr, 62
add a string of wchar_t characters to a curses
window and advance cursor —
curs_addwstr
addnwstr, 64
add a wchar_t character (with attributes) to a
curses window and advance cursor —
curs_addwch
addwchstr, 62
add a string of wchar_t characters (and attributes)
to a curses window — curs_addwchstr
addwchnstr, 62
add a wchar_t character (with attributes) to a
curses window and advance cursor —
curs_addwch
addwchstr, 62
add a string of wchar_t characters to a curses
window and advance cursor —
curs_addwstr
addwstr, 64
add a wchar_t character (with attributes) to a
curses window and advance cursor —
curs_addwch
echowchar, 59
add a string of wchar_t characters to a curses
window and advance cursor —
curs_addwstr
mvaddnwstr, 64
add a wchar_t character (with attributes) to a
curses window and advance cursor —
curs_addwch
addwchstr, 59
add string of wchar_t characters (and attributes)
to a curses window — curs_addwchstr
mvaddwchnstr, 62
mvaddwchstr, 62
add a string of wchar_t characters to a curses
window and advance cursor —
curs_addwstr
mvwaddwstr, 64
add a wchar_t character (with attributes) to a
curses window and advance cursor —
curs_addwch
mvwaddwch, 59
add string of wchar_t characters (and attributes)
to a curses window — curs_addwchstr
mvwaddwchnstr, 62
mvwaddwchstr, 62
add a string of wchar_t characters to a curses
window and advance cursor —
curs_addwstr
mvwaddwstr, 64
waddnwstr, 64
add a wchar_t character (with attributes) to a
  curses window and advance cursor —
  curs_addwch
  waddwch, 59
add string of wchar_t characters (and attributes) to a curses window — curs_addwchstr
  waddwchnstr, 62
  waddwchstr, 62
add a string of wchar_t characters to a curses window and advance cursor —
  curs_addwchstr
  waddwstr, 64
add a wchar_t character (with attributes) to a curses window and advance cursor —
  curs_addwch
  wehowwchar, 59
add a character (with rendition) to a window
  — addch, 18
  — mvaddch, 18
  — mvwaddch, 18
  — waddch, 18
add a complex character (with rendition) to a window
  — add_wch, 25
  — mvadd_wch, 25
  — mvwadd_wch, 25
  — wadd_wch, 25
add a complex character and refresh window
  — echo_wchar, 184
  — wecho_wchar, 184
add a multi-byte character string (without rendition) to a window
  — addnstr, 21
  — addstr, 21
  — mvaddnstr, 21
  — mvaddstr, 21
  — mvwaddnstr, 21
  — mvwaddstr, 21
  — waddnstr, 21
  — waddstr, 21
add a single-byte border to a window
  — border, 38
  — box, 38
  — wborders, 38
add a single-byte character and refresh window
  — echochar, 183
  — wechochar, 183
add a wide-character string to a window
  — addnwstr, 23
  — addwstr, 23
  — mvaddnwstr, 23
  — mvaddwstr, 23
  — mvwaddnwstr, 23
  — mvwaddwstr, 23
  — waddnwstr, 23
  — waddwstr, 23
add character and refresh window
  — pecho_wchar, 323
  — pechochar, 323
add_wch — add a complex character (with rendition) to a window, 25
add_wchstr — copy a string of complex characters (with renditions) to a window, 27
add_wchstr — copy a string of complex characters (with renditions) to a window, 27
addch — add a character (with rendition) to a window, 18
addchnstr — copy a character string (with renditions) to a window, 20
addchstr — copy a character string (with renditions) to a window, 20
addnstr — add a multi-byte character string (without rendition) to a window, 21
addnwstr — add a string of wchar_t characters to a curses window and advance cursor, 64
addnwstr — add a wide-character string to a window, 23
addstr — add a multi-byte character string (without rendition) to a window, 21
addwch — add a wchar_t character (with attributes) to a curses window and advance cursor, 59
addwchnstr — add string of wchar_t characters (and attributes) to a curses window, 62
addwchstr — add string of wchar_t characters (and attributes) to a curses window, 62
addwstr — add a string of wchar_t characters to a curses window and advance cursor, 64
addwstr — add a wide-character string to a window, 23
adjcurspos — moving the cursor by character, 65
ALE curses library, See curses library
apply padding information and output string
— putp, 327
— tputs, 327
arc — graphics interface, 324
attr_get — control window attributes, 29
attr_off — control window attributes, 29
attr_on — control window attributes, 29
attr_set — control window attributes, 29
attroff — change foreground window
attributes, 31
attroff — curses character and window attribute
control routines, 66
attron — change foreground window
attributes, 31
attron — curse a character and window attribute
control routines, 66
attrset — change foreground window
attributes, 31
attrset — curses character and window attribute
control routines, 66
B
baudrate — return terminal baud rate, 32
beep — activate audio-visual alarm, 33
bkgd — set or get the background character
(and rendition) of window, 34
bkgdset — set or get the background character
(and rendition) of window, 34
bkgrnd — set or get the background character
(and rendition) of window using a complex
character, 36
bkgrndset — set or get the background
character (and rendition) of window using a
complex character, 36
border — add a single-byte border to a
window, 38
border_set — use complex characters (and
renditions) to draw borders, 40
box — add a single-byte border to a
window, 38
box — graphics interface, 324
box_set — use complex characters (and
renditions) to draw borders, 40
C
call refresh on changes to window —
immedok, 238
can_change_color — manipulate color
information, 42
cbreak — set input mode controls, 45
change foreground window attributes
— attroff, 31
— attron, 31
— attrset, 31
— wattroff, 31
— wattron, 31
— wattrset, 31
change the rendition of characters in a window
— chgat, 46
— mvchgat, 46
— mvwchgat, 46
— wchgat, 46
color character based forms package
— forms, 214
color character based menus package
— menus, 290
color character based panels package
— panels, 316
check for type-ahead characters —
typeahead, 346
chgat — change the rendition of characters in a
window, 46
circle — graphics interface, 324
clear — clear a window, 47
clear a window
— clear, 47
— erase, 47
— wclear, 47
— werase, 47
clear to the end of a line
— clrtoeol, 51
— wclrtoeol, 51
clear to the end of a window
— clrtobot, 50
— wclrtobot, 50
clearok — set terminal output controls, 48
closepl — graphics interface, 324
closevt — graphics interface, 324
clrtobot — clear to the end of a window, 50
clrtoeol — clear to the end of a line, 51
copy a character string (with renditions) to a window (continued)
  — addchstr, 20
  — mvaddchnstr, 20
  — mvaddchstr, 20
  — mvwaddchnstr, 20
  — mvwaddchstr, 20
  — waddchstr, 20
  — waddchstr, 20

copy a string of complex characters (with renditions) to a window
  — add_wchnstr, 27
  — add_wchstr, 27
  — mvadd_wchnstr, 27
  — mvadd_wchstr, 27
  — mvwadd_wchnstr, 27
  — mvwadd_wchstr, 27
  — wadd_wchnstr, 27
  — wadd_wchstr, 27

create a new window or subwindow
  — derwin, 178
  — newwin, 178
  — subwin, 178

create or refresh a pad or subpad
  — newpad, 304
  — pnoutrefresh, 304
  — prefresh, 304
  — subpad, 304

CRT handling and optimization package
  — curses, 79

cur_term — current terminal information, 169
  — cur_term, 169
  — curscr, 76
  — curs_addwch — add a wchar_t character (with attributes) to a curses window and advance cursor, 59
  — curs_addwchstr — add string of wchar_t characters (and attributes) to a curses window, 62
  — curs_addwstr — add a string of wchar_t characters to a curses window and advance cursor, 64
  — curs_alecompat — moving the cursor by character,
curses character and window attribute control routines (continued)
  — wstandend, 66
  — wstandout, 66

curses color manipulation routines
  — can_change_colors, 73
  — color_content, 73
  — curs_color, 73
  — has_colors, 73
  — init_color, 73
  — init_pair, 73
  — pair_content, 73
  — start_color, 73

curses cursor and window coordinates
  — curs_getyx, 117
  — getbegyx, 117
  — getmaxyx, 117
  — getparyx, 117
  — getyx, 117

curses environment query routines
  — baudrate, 154
  — curs_termattrs, 154
  — erasechar, 154
  — has_ic, 154
  — has_il, 154
  — killchar, 154
  — longname, 154
  — termattrs, 154
  — termname, 154

curses interfaces to termcap library
  — curs_termcap, 156
  — tgetent, 156
  — tgetflag, 156
  — tgetnum, 156
  — tgetstr, 156
  — tgoto, 156
  — tputs, 156

curses interfaces to terminfo database
  See also form library, menu library, or panel library
  — curs_touch, 162
  — is_linetouched, 162
  — is_wintouched, 162
  — touchline, 162
  — touchwin, 162
  — untouchwin, 162
  — wtouchln, 162

curses refresh control routines
  — curs_scr_dump, 148
  — scr_dump, 148
  — scr_init, 148
  — scr_restore, 148
  — scr_set, 148

curses screen initialization and manipulation routines
  — adjcurspos, 65
  — curs_alecompat, 65
  — moveprevch, 65
  — wadcurspos, 65
  — wmovenextch, 65
  — wmoveprevch, 65

See also form library, menu library, or panel library
— curs_termcap, 156
— tgetent, 156
— tgetflag, 156
— tgetnum, 156
— tgetstr, 156
— tgoto, 156
— tputs, 156

See also form library, menu library, or panel library
— curs_touch, 162
— is_linetouched, 162
— is_wintouched, 162
— touchline, 162
— touchwin, 162
— untouchwin, 162
— wtouchln, 162
curses screen initialization and manipulation routines (continued)
— initscr, 120
— isendwin, 120
— newterm, 120
— set_term, 120

Curses soft label routines
— curs_silk, 152
— silk_attroff, 152
— silk_attron, 152
— silk_attrset, 152
— silk_clear, 152
— silk_init, 152
— silk_label, 152
— silk_noutrefresh, 152
— silk_refresh, 152
— silk_restore, 152
— silk_set, 152
— silk_touch, 152

Curses terminal input option control routines
— cbreak, 122
— curs_inopts, 122
— echo, 122
— halfdelay, 122
— intrflush, 122
— keypad, 122
— meta, 122
— nobreak, 122
— nodelay, 122
— noecho, 122
— noqiflush, 122
— noraw, 122
— notimeout, 122
— qiflush, 122
— raw, 122
— timeout, 122
— typeahead, 122
— wtimeout, 122

Curses window, add character and advance cursor
— addch, 54
— curs_addch, 54
— echochar, 54
— mvwaddch, 54
— waddch, 54
— wechochar, 54

Curses window, add string of characters
— addchnstr, 57
— addchstr, 57
— curs_addchstr, 57
— mvaddchnstr, 57
— mvaddchstr, 57
— mvwaddchnstr, 57
— mvwaddchstr, 57
— waddchnstr, 57
— waddchstr, 57

Curses window, add string of characters and advance cursor
— addnstr, 58
— addstr, 58
— curs_addnstr, 58
— curs_addstr, 58
— mvaddnstr, 58
— mvaddstr, 58
— mvwaddnstr, 58
— mvwaddstr, 58
— waddnstr, 58
— waddstr, 58
curses window, clear all or part
— clear, 72
— clrtoeol, 72
— curs_clear, 72
— erase, 72
— wclear, 72
— wcrltob, 72
— wcrlteol, 72
— werase, 72

curses window, convert formatted input
— curs_scanw, 147
— mvscanw, 147
— mvwscanw, 147
— scanw, 147
— vwscanw, 147
— wscanw, 147

curses window, delete and insert lines
— curs_deleteln, 78
— deleteln, 78
— insdelln, 78
— insertln, 78
— wdeleteln, 78
— winsdelln, 78
— wininsertn, 78

curses window, delete character under cursor
— curs_delch, 77
— delch, 77
— mvdelch, 77
— mwdelch, 77
— wdelch, 77

curses window, get character and its attributes
— curs_inch, 118
— inch, 118
— mvinch, 118
— mvwinch, 118
— winch, 118

curses window, get string of characters
— curs_inchstr, 119
— curs_instr, 128
— inchstr, 119
— inchstr, 119
— innstr, 128
— instr, 128
— mvinchstr, 119
— mvinsstr, 128
— mvinsstr, 128
— winchstr, 119
— winchstr, 128
— winstr, 128

(continued)
curses window, get string of characters
— mvinstr, 128
— mwwinchstr, 119
— mvwinchstr, 119
— mvwinchstr, 128
— mvwinstr, 128
— winchln, 119
— winchstr, 128
— winstr, 128

curses window, insert character before character under cursor
— curs_insch, 125
— insch, 125
— mvinsch, 125
— mvwinsch, 125
— winsch, 125

curses window, insert string before character under cursor
— curs_instr, 126
— insnstr, 126
— instr, 126
— mvinsnstr, 126
— mvinsstr, 126
— mvwinsnstr, 126
— mvwinsstr, 126
— winsnstr, 126
— winsstr, 126

curses window, scroll
— curs_scroll, 150
— scrl, 150
— scroll, 150
— wscrl, 150

curses window background manipulation routines
— bkgd, 69
— bkgdset, 69
— curs_bkgd, 69
— wbkgd, 69
— wbkgdset, 69

curses window cursor
— curs_move, 137
— move, 137
— wmove, 137

curses windows, create
— curs_window, 166
curses windows, create (continued)
— delwin, 166
— derwin, 166
— dupwin, 166
— mvderwin, 166
— mvwin, 166
— newwin, 166
— subwin, 166
— syncok, 166
— wcursyncup, 166
— wsyncdown, 166
— wsyncup, 166
curses windows, overlap and manipulate
— copywin, 141
— curs_overlay, 141
— overlay, 141
— overwrite, 141
curses windows, print formatted output
— curs_printw, 144
— mvprintw, 144
— mvwprintw, 144
— printw, 144
— vwprintw, 144
— wprintw, 144
curses windows and lines, refresh
— curs_refresh, 145
— doupdate, 145
— redrawwin, 145
— refresh, 145
— wnoutrefresh, 145
— wredrawln, 145
— wrefresh, 145

def prog mode — save/restore terminal modes, 170
def shell mode — save/restore terminal modes, 170
default window — stdscr, 339
del_curterm — free space pointed to by terminal, 173
delay_output — delays output, 171
delays output — delay_output, 171
delch — remove a character, 172
delete a window — delwin, 177
deleteln — remove a line, 175
delwin — delete a window, 177
derwin — create a new window or subwindow, 178
determine insert/delete character/line capability
— has_ic, 234
— has_il, 234
disable use of certain terminal capabilities — filter, 187
discard type-ahead characters — flushinp, 188
doupdate — refresh windows and lines, 180
duplicate a window — dupwin, 181
dupwin — duplicate a window, 181

E
echo — enable/disable terminal echo, 182
echo wchar — add a complex character and refresh window, 184
echochar — add a single-byte character and refresh window, 183
echowchar — add a wchar_t character (with attributes) to a curses window and advance cursor, 59
emulate the termcap database
— tgetent, 343
— tgetflag, 343
— tgetnum, 343
— tgetstr, 343
— tgoto, 343
enable/disable half-delay mode — halfdelay, 233
enable/disable hardware insert-character and delete-character features — idcok, 237
enable/disable keypad handling — keypad, 262
enable/disable meta keys — meta, 296
enable/disable newline control
— nl, 306
— nonl, 306
enable/disable terminal echo
— echo, 182
— noecho, 182
endwin — restore initial terminal environment, 185
erase — clear a window, 47
erase — graphics interface, 324
erasechar — return current ERASE or KILL characters, 186
erasewchar — return current ERASE or KILL characters, 186

F
filter — disable use of certain terminal capabilities, 187
flash — activate audio-visual alarm, 33
enable or disable flush on interrupt — intrflush, 255
fluship — discard type-ahead characters, 188
form library. See also curses library
forms — character based forms package, 214
forms, application-specific routines
— field_init, 206
— field_term, 206
— form_hook, 206
— form_init, 206
— form_term, 206
— set_field_init, 206
— set_field_term, 206
— set_form_init, 206
— set_form_term, 206
forms, associate application data
— field_userptr, 204
— form_field_userptr, 204
— form_userptr, 218
— set_field_userptr, 204
— set_form_userptr, 218
forms, command processor
— form_driver, 191
forms, connect fields
— field_count, 194
— form_field, 194
— form_fields, 194
— move_field, 194
— set_form_fields, 194
forms, create and destroy
— form_new, 208
— free_form, 208
— new_form, 208
forms, format general appearance
— field_just, 198
— form_field_just, 198
— set_field_just, 198
forms, format general display attributes
— field_back, 195
— field_fore, 195
— field_pad, 195
— form_field_attributes, 195
— set_field_back, 195
— set_field_fore, 195
— set_field_pad, 195
forms, set current page and field
— current_field, 211
— field_index, 211
— form_page, 211
— set_current_field, 211
— set_form_page, 211
forms, write/erase from associated subwindows
— field_buffer, 196
— field_status, 196
— form_field_buffer, 196
— set_field_buffer, 196
— set_field_status, 196
— set_max_field, 196
forms field attributes, set and get
— field_buffer, 196
— field_status, 196
— form_field_buffer, 196
— set_field_buffer, 196
— set_field_status, 196
— set_max_field, 196
forms field characteristics
— dynamic_field_info, 197
— field_info, 197
— form_field_info, 197
forms field data type validation
— field_arg, 205
— field_type, 205
— form_field_validation, 205
— set_field_type, 205
forms field option routines
— field_opts, 200
— field_opts_off, 200
— field_opts_on, 200
forms field option routines (continued)
  — form_field_opts, 200
  — set_field_opts, 200
forms fields, create and destroy
  — dup_field, 199
  — form_field_new, 199
  — free_field, 199
  — link_field, 199
  — new_field, 199
forms fieldtype routines
  — form_fieldtype, 202
  — free_fieldtype, 202
  — link_fieldtype, 202
  — new_fieldtype, 202
  — set_fieldtype_arg, 202
  — set_fieldtype_choice, 202
forms option routines
  — form_opts, 210
  — form_opts_off, 210
  — form_opts_on, 210
  — set_form_opts, 210
forms pagination
  — form_new_page, 209
  — new_page, 209
  — set_new_page, 209
forms window and subwindow association routines
  — form_sub, 219
  — form_win, 219
  — scale_form, 219
  — set_form_sub, 219
  — set_form_win, 219
forms window cursor, position
  — form_cursor, 189
  — pos_form_cursor, 189
free space pointed to by terminal
  — del_curterm, 173
  — restartterm, 173
  — set_curterm, 173
  — setterm, 173
  — setupterm, 173

G
generate printable representation of a character
  — unctrl, 347

generate printable representation of a wide character — wunctrl, 356
generate printable representation of a wide character — wunctrl, 356
generate printable representation of a wide character — wunctrl, 356
get a string of wchar_t characters and attributes from a curses window —
curs_inwchstr, 133
get a string of wchar_t characters from a curses window — curs_inwstr, 134
get a wchar_t character and its attributes from a
curses window — curs_inwch, 132
get supported terminal video attributes —
termattrs, 341
get supported terminal video attributes —
term_attrs, 341
get wchar_t character strings from curses
terminal keyboard — curs_getwstr, 116
getwstr, 116
get wchar_t character strings from curses
terminal keyboard — curs_getwstr
getwstr, 116
get a string of wchar_t characters from a curses window —
curs_inwch
inwch, 132
get a wchar_t character and its attributes from a
curses window — curs_inwch
inwch, 132
get a string of wchar_t characters (and attributes) from a curses window —
curs_inwchstr
inwchnstr, 133
inwchstr, 133
get a string of wchar_t characters from a curses window — curs_inwstr
inwstr, 134
get wchar_t character strings from curses
terminal keyboard — curs_getwstr
mvgetnwstr, 116
mvgetwstr, 116
get a string of wchar_t characters from a curses window —
curs_inwch
mvinwch, 132
get a wchar_t character and its attributes from a
curses window — curs_inwch
mvinwch, 132
get a string of wchar_t characters (and attributes) from a curses window —
curs_inwchstr
mvinwchnstr, 133
get a string of wchar_t characters (and attributes) from a curses window —
curs_inwchstr (continued)
  mvwinwchstr, 133
get a string of wchar_t characters from a curses window — curs_inwstr
  mvwinwstr, 134
get wchar_t character strings from curses terminal keyboard —
curs_getwstr
  mvwgetwstr, 116
  mvwgetwstr, 116
get a string of wchar_t characters from a curses window — curs_inwstr
  mvwinnwstr, 134
get a wchar_t character and its attributes from a curses window — curs_inwch
  mvwinwch, 132
get a string of wchar_t characters (and attributes) from a curses window —
curs_inwchstr
  mvwinwchinwchstr, 133
  mvwinwchinwchstr, 133
get a string of wchar_t characters from a curses window — curs_inwstr
  mvwinwstr, 134
get wchar_t character strings from curses terminal keyboard — curs_getwstr
  wgetwstr, 116
  wgetwstr, 116
get a string of wchar_t characters from a curses window — curs_inwstr
  winwstr, 134
get a wchar_t character and its attributes from a curses window — curs_inwch
  winwch, 132
get a string of wchar_t characters (and attributes) from a curses window —
curs_inwchstr
  winwchinwchstr, 133
  winwchinwchstr, 133
get a string of wchar_t characters from a curses window — curs_inwstr
  winwstr, 134
get (or push back) wchar_t characters from curses terminal keyboard (continued)
  — mvgetwch, 111
  — mvwgetwch, 111
  — ungetwch, 111
  — wgetwch, 111
get a multibyte character string from terminal
  — getnstr, 227
  — getstr, 227
  — mvgetnstr, 227
  — mvgetstr, 227
  — mvwgetnstr, 227
  — mvwgetstr, 227
  — wgetnstr, 227
  — wgetstr, 227
get a single-byte character from terminal
  — getch, 222
  — mvgetch, 222
  — mvwgetch, 222
  — wgetch, 222
get a wide character from terminal
  — get_wch, 230
  — mvget_wch, 230
  — mvwget_wch, 230
  — wget_wch, 230
get a wide character string (with rendition) from a cchar_t — getchar, 221
get a wide character string from terminal
  — get_wstr, 229
  — getn_wstr, 229
  — mvget_wstr, 229
  — mvgetn_wstr, 229
  — mvwget_wstr, 229
  — mvwgetn_wstr, 229
  — wget_wstr, 229
  — wgetn_wstr, 229
get cursor or window coordinates
  — getbegyx, 220
  — getmaxyx, 220
  — getparyx, 220
  — getyx, 220
get_wch — get a wide character from terminal, 230
get_wstr — get a wide character string from terminal, 229
getbegyx — get cursor or window coordinates, 220
getbkgd — set or get the background character (and rendition) of window, 34
getbkgrnd — set or get the background character (and rendition) of window using a complex character, 36
getcchar — get a wide character string (with rendition) from a cchar_t, 221
getch — get a single-byte character from terminal, 222
getmaxyx — get cursor or window coordinates, 220
getn_wstr — get a wide character string from terminal, 229
getnstr — get a multibyte character string from terminal, 227
getnwstr — get wchar_t character strings from curses terminal keyboard, 116
getparyx — get cursor or window coordinates, 220
getstr — get a multibyte character string from terminal, 227
getwch — get (or push back) wchar_t characters from curses terminal keyboard, 111
getwin — read a window from, and write a window to, a file, 232
getwstr — get wchar_t character strings from curses terminal keyboard, 116
getyx — get cursor or window coordinates, 220

graphics interface
— arc, 324
— box, 324
— circle, 324
— closepl, 324
— closevt, 324
— cont, 324
— erase, 324
— label, 324
— line, 324
— linmod, 324
— move, 324
— openpl, 324
— openvt, 324
— plot, 324
— point, 324
— space, 324

H
halfdelay — enable/disable half-delay mode, 233
has_colors — manipulate color information, 42
has_ic — determine insert/delete character/line capability, 234
has_il — determine insert/delete character/line capability, 234
hline — use single-byte characters (and renditions) to draw lines, 235
hline_set — use complex characters (and renditions) to draw lines, 236

I
idcok — enable/disable hardware insert-character and delete-character features, 237
idlok — set terminal output controls, 48
immedok — call refresh on changes to window, 238
in_wch — retrieve a complex character (with rendition), 256
in_wchnstr — retrieve complex character string (with rendition), 257
in_wchstr — retrieve complex character string (with rendition), 257
inch — return a single-byte character (with rendition), 239
inchstr — retrieve a single-byte character string (with rendition), 240
inchstr — retrieve a single-byte character string (with rendition), 240
init_color — manipulate color information, 42
init_pair — manipulate color information, 42
initscr — screen initialization functions, 242
innstr — retrieve a multibyte character string (without rendition), 243
innwstr — get a string of wchar_t characters from a curses window, 134
innwstr — retrieve a wide character string (without rendition), 245
ins_nwstr — insert a wide character string, 252
ins_wch — insert a complex character, 254
ins_wstr — insert a wide character string, 252
insch — insert a character, 247
insdeln — insert/delete lines to/from the window, 248
insert a wchar_t character before the character under the cursor in a curses window —
curs_inswch, 129
insnwstr, 130
insert a wchar_t string before character under the
cursor in a curses window —
curs_inswstr, 130
inswch, 129
insert wchar_t string before character under the
cursor in a curses window — curs_inswstr
inswstr, 130
mvinsnwstr, 130
insert a wchar_t character before the character under the cursor in a curses window —
curs_inswch
mvinswch, 129
insert wchar_t string before character under the
cursor in a curses window — curs_inswstr
mvinswstr, 130
mvwinsnwstr, 130
insert a wchar_t character before the character under the cursor in a curses window —
curs_inswch
mvwinswch, 129
insert wchar_t string before character under the
cursor in a curses window — curs_inswstr
mvwinswstr, 130
mvwinsnwstr, 130
insert a wchar_t character before the character under the cursor in a curses window —
curs_inswch
mvwinsch, 129
insert wchar_t string before character under the
cursor in a curses window — curs_inswstr
mvwinswstr, 130
mvwinsnwstr, 130
insert a wchar_t character before the character under the cursor in a curses window —
curs_inswch
mvwinsch, 129
insert wchar_t string before character under the
cursor in a curses window — curs_inswstr
mvwinswstr, 130
mvwinsnwstr, 130
insert a character
— insch, 247
— mvinsch, 247
— mvwinsch, 247
— winsch, 247
insert a complex character
— ins_wch, 254
— mvins_wch, 254
— mvwins_wch, 254
— wins_wch, 254
insert a line in a window
— insertln, 249
— winsertln, 249
insert a multibyte character string
— insnstr, 250
— insstr, 250
— mvinsnstr, 250
— mvinsstr, 250
— mvwinsnstr, 250
— mvwinsstr, 250
— winsnstr, 250
— winsstr, 250
insert a wide character string
— ins_nwstr, 252
— ins_wstr, 252
— mvinsnstr, 252
— mvinsstr, 252
— mvwinsnstr, 252
— mvwinsstr, 252
— wins_nwstr, 252
— wins_wstr, 252
insert/delete lines to/from the window
— insdeln, 248
— winsdeln, 248
insertln — insert a line in a window, 249
insnstr — insert a multibyte character string, 250
insnwstr — insert wchar_t string before character under the cursor in a curses window, 130
insstr — insert a multibyte character string, 250
instr — retrieve a multibyte character string (without rendition), 243
inswch — insert a wchar_t character before the character under the cursor in a curses window, 129
inswstr — insert wchar_t string before character under the cursor in a curses window, 130
intrflush — enable or disable flush on interrupt, 255
introduction and overview of X/Open Curses — curses, 94
inwch — get a wchar_t character and its attributes from a curses window, 132
inwchnstr — get a string of wchar_t characters (and attributes) from a curses window, 133
inwchstr — get a string of wchar_t characters (and attributes) from a curses window, 133
inwstr — get a string of wchar_t characters from a curses window, 134
inwstr — retrieve a wide character string (without rendition), 245
is_linetouched — control window refresh, 259
is_wintouched — control window refresh, 259
isendwin — restore initial terminal environment, 185

K
key_name — return character string used as key name, 261
keyname — return character string used as key name, 261
keypad — enable/disable keypad handling, 262
killchar — return current ERASE or KILL characters, 186
killwchar — return current ERASE or KILL characters, 186

L
label — graphics interface, 324
leaveok — set terminal output controls, 48
line — graphics interface, 324
LINES — number of lines on terminal screen, 266
linmod — graphics interface, 324
longname — return full terminal type name, 267

M
manipulate color information —
can_change_color, 42
manipulate color information —
color_content, 42
manipulate color information —
COLOR_PAIR, 42
manipulate color information —
COLOR_PAIRS, 42
manipulate color information —
COLORS, 42
manipulate color information —
has_colors, 42
manipulate color information —
init_color, 42
manipulate color information —
init_pair, 42
manipulate color information —
pair_content, 42
manipulate color information —
PAIR_NUMBER, 42
manipulate color information —
start_color, 42
map area of parent window to subwindow —
mvderwin, 299
menu library, See also curses library
menus — character based menus package, 290
menus, application-specific routines
— item_init, 274
— item_term, 274
— menu_hook, 274
— menu_init, 274
— menu_term, 274
— set_item_init, 274
— set_item_term, 274
— set_menu_init, 274
— set_menu_term, 274
menus, associate application data
— menu_userptr, 294
— set_menu_userptr, 294
menus, create and destroy
— free_menu, 286
— menu_new, 286
— new_menu, 286
menus, rows and columns
— menu_format, 273
— set_menu_format, 273
menus cursor
— menu_cursor, 270
— pos_menu_cursor, 270
menus display attributes
— menu_attributes, 268
menus display attributes (continued)
  — menu_back, 268
  — menu_fore, 268
  — menu_grey, 268
  — menu_pad, 268
  — set_menu_back, 268
  — set_menu_fore, 268
  — set_menu_grey, 268
  — set_menu_pad, 268
menus from associated subwindows, write/erase
  — menu_post, 289
  — post_menu, 289
  — unpost_menu, 289
menus item, visibility
  — item_visible, 284
  — menu_item_visible, 284
menus item name and description
  — item_description, 278
  — item_name, 278
  — menu_item_name, 278
menus item options routines
  — item_opts, 280
  — item_opts_off, 280
  — item_opts_on, 280
  — menu_item_opts, 280
  — set_item_opts, 280
menus item values, set and get
  — item_value, 283
  — menu_item_value, 283
  — set_item_value, 283
menus items, associate application data
  — item_userptr, 282
  — menu_item_userptr, 282
  — set_item_userptr, 282
menus items, connect and disconnect
  — item_count, 281
  — menu_items, 281
  — set_menu_items, 281
menus items, create and destroy
  — free_item, 279
  — menu_item_new, 279
  — new_item, 279
menus items, get and set
  — current_item, 276
  — item_index, 276
  — menu_item_current, 276
menus items, get and set (continued)
  — set_current_item, 276
  — set_top_row, 276
  — top_row, 276
menus mark string routines
  — menu_mark, 285
  — set_menu_mark, 285
menus options routines
  — menu_opts, 287
  — menuOpts_off, 287
  — menu_opts_on, 287
  — set_menu_opts, 287
menus pattern match buffer
  — menu_pattern, 288
  — set_menu_pattern, 288
menus subsystem, command processor
  — menu_driver, 271
menus window and subwindow association routines
  — menu_sub, 295
  — menu_win, 295
  — scale_menu, 295
  — set_menu_sub, 295
  — set_menu_win, 295
meta — enable/disable meta keys, 296
move — graphics interface, 324
move — move cursor in window, 297
move cursor in window
  — move, 297
  — wmove, 297
move the cursor — mvcur, 298
move window — mvwin, 302
movenextch — moving the cursor by character, 65
moveprevch — moving the cursor by character, 65
mvadd_wch — add a complex character (with rendition) to a window, 25
mvaddwchnstr — copy a string of complex characters (with renditions) to a window, 27
mvaddwchstr — copy a string of complex characters (with renditions) to a window, 27
mvaddch — add a character (with rendition) to a window, 18
mvaddchnstr — copy a character string (with renditions) to a window, 20
mvaddchstr — copy a character string (with renditions) to a window, 20
mvaddnstr — add a multi-byte character string (without rendition) to a window, 21
mvaddnwstr — add a string of wchar_t characters to a curses window and advance cursor, 64
mvaddnstr — add a wide-character string to a window, 23
mvaddstr — add a multi-byte character string (without rendition) to a window, 21
mvaddwch — add a wchar_t character (with attributes) to a curses window and advance cursor, 59
mvaddwchnstr — add string of wchar_t characters (and attributes) to a curses window, 62
mvaddwcharstr — add string of wchar_t characters (and attributes) to a curses window, 62
mvaddwstr — add a wide-character string to a window, 23
mvchgat — change the rendition of characters in a window, 46
mvcur — move the cursor, 298
mvdelch — remove a character, 172
mvderwin — map area of parent window to subwindow, 299
mvget_wch — get a wide character from terminal, 230
mvget_wstr — get a wide character string from terminal, 229
mvgetch — get a single-byte character from terminal, 222
mvgetn_wstr — get a wide character string from terminal, 229
mvgetnstr — get a multibyte character string from terminal, 227
mvgetnstr — get wchar_t character strings from curses terminal keyboard, 116
mvgetstr — get a multibyte character string from terminal, 227
mvgetwch — get (or push back) wchar_t characters from curses terminal keyboard, 111
mvgetwstr — get wchar_t character strings from curses terminal keyboard, 116
mvhline — use single-byte characters (and renditions) to draw lines, 235
mvhline_set — use complex characters (and renditions) to draw lines, 236
mvwinch — retrieve a complex character (with rendition), 256
mvwch_str — retrieve complex character string (with rendition), 257
mvwchstr — retrieve complex character string (with rendition), 257
mvwinch — return a single-byte character (with rendition), 239
mvwchnstr — retrieve a single-byte character string (with rendition), 240
mvwchstr — retrieve a single-byte character string (with rendition), 240
mvwinn — retrieve a multibyte character string (without rendition), 243
mvwinnstr — get a string of wchar_t characters from a curses window, 134
mvwinnstr — retrieve a wide character string (without rendition), 245
mvwusrw — insert a wide character string, 252
mvwusrw — insert a complex character, 254
mvwusrw — insert a wide character string, 252
mvwins — insert a character, 247
mvwinnstr — insert a multibyte character string, 250
mvwinnwstr — insert wchar_t character strings before character under the cursor in a curses window, 130
mvwinnwstr — insert a multibyte character string, 250
mvwinnstr — retrieve a multibyte character string (without rendition), 243
mvwinnch — insert a wchar_t character before the character under the cursor in a curses window, 129
mvinswstr — insert wchar_t string before character under the cursor in a curses window, 130
mvinwch — get a wchar_t character and its attributes from a curses window, 132
mvinwchnstr — get a string of wchar_t characters (and attributes) from a curses window, 133
mvinwchstr — get a string of wchar_t characters (and attributes) from a curses window, 133
mvinwstr — get a string of wchar_t characters from a curses window, 134
mvinwstr — retrieve a wide character string (without rendition), 245
mvprintw — print formatted output window, 300
mvscanw — convert formatted input from a window, 301
mvvline — use single-byte characters (and renditions) to draw lines, 235
mvvline_set — use complex characters (and renditions) to draw lines, 236
mvwaddwch — add a wchar_t character (with attributes) to a curses window, 62
mvwaddwchstr — add string of wchar_t characters (and attributes) to a curses window, 62
mvwaddwchstr — add string of wchar_t characters (and attributes) to a curses window, 62
mvwaddwstr — add a string of wchar_t characters to a curses window and advance cursor, 64
mvwaddwstr — add a wide-character string to a window, 23
mvwchgat — change the rendition of characters in a window, 46
mvwdelch — remove a character, 172
mvwget_wch — get a wide character from terminal, 230
mvwget_wstr — get a wide character string from terminal, 229
mvwgetch — get a single-byte character from terminal, 222
mvwgetn_wstr — get a wide character string from terminal, 229
mvwgetnstr — get a multibyte character string from terminal, 227
mvwgetwch — get (or push back) wchar_t characters from curses terminal keyboard, 111
mvwgetwstr — get wchar_t character strings from curses terminal keyboard, 116
mvwhline — use single-byte characters (and renditions) to draw lines, 235
mvwhline_set — use complex characters (and renditions) to draw lines, 236
mvwinc — move window, 302
mvwin — move window, 302
mvwin_wch — retrieve a complex character (with rendition), 256
mvwin_wchnstr — retrieve complex character string (with rendition), 257
mvwin_wchstr — retrieve complex character string (with rendition), 257
mvwin — return a single-byte character (with rendition), 239
mvwinchnstr — retrieve a single-byte character string (with rendition), 240
mvwinchstr — retrieve a single-byte character string (with rendition), 240
mvwinnstr — retrieve a multibyte character string (without rendition), 243
mvwinnwstr — get a string of wchar_t characters from a curses window, 134
mvwinnwstr — retrieve a wide character string (without rendition), 245
mvwins_nstr — insert a wide character string, 252
mvwins_nwstr — insert a wide character string, 252
mvwins_wch — insert a complex character, 254
mvwinsch — insert a character, 247
mvwinsnstr — insert a multibyte character string, 250
mvwinsnwstr — insert wchar_t string before character under the cursor in a curses window, 130
mvwinsstr — insert a multibyte character string, 250
mvwinstr — retrieve a multibyte character string (without rendition), 243
mvwinswch — insert a wchar_t character before the character under the cursor in a curses window, 129
mvwinswstr — insert wchar_t string before character under the cursor in a curses window, 130
mvwinwch — get a wchar_t character and its attributes from a curses window, 132
mvwinwchnstr — get a string of wchar_t characters (and attributes) from a curses window, 133
mvwinwchstr — get a string of wchar_t characters (and attributes) from a curses window, 133
mvwinwstr — get a string of wchar_t characters from a curses window, 134
mvwinwstr — retrieve a wide character string (without rendition), 245
mvwprintw — print formatted output window, 300
mvwscanw — convert formatted input from a window, 301
mvwvline — use single-byte characters (and renditions) to draw lines, 235
mvwvline_set — use complex characters (and renditions) to draw lines, 236
mwwaddnstr — add a multi-byte character string (without rendition) to a window, 21

N

napms — sleep process for a specified length of time, 303
newpad — create and display curses pads, 142
newpad — create or refresh a pad or subpad, 304
newterm — screen initialization functions, 242
newwin — create a new window or subwindow, 178
nl — enable/disable newline control, 306
nocbreak — set input mode controls, 45
nodelay — set blocking or non-blocking read, 307
noecho — enable/disable terminal echo, 182
nonl — enable/disable newline control, 306
noqiflush — control flush of input and output on interrupt, 308
noraw — set input mode controls, 45
notimeout — set timed blocking or non-blocking read, 309
number of columns on terminal screen — COLS, 52
number of lines on terminal screen — LINES, 266

O
openpl — graphics interface, 324
openvt — graphics interface, 324
output attributes to the terminal — vidattr, 350
output attributes to the terminal — vid_attr, 350
output attributes to the terminal — vidputs, 350

Index 375
output attributes to the terminal —
  vid_puts, 350
overlay — copy overlapped windows, 310
overlay or overwrite any portion of window —
  copywin, 53
overwrite — copy overlapped windows, 310

Pair content — manipulate color information, 42
PAIR_NUMBER — manipulate color information, 42
panel library, See also curses library
panels — character based panels package, 316
panels, create and destroy
  — del_panel, 315
  — new_panel, 315
  — panel_new, 315
panels deck manipulation routines
  — bottom_panel, 319
  — hide_panel, 318
  — panel_hidden, 318
  — panel_show, 318
  — panel_top, 319
  — show_panel, 318
  — top_panel, 319
panels deck traversal primitives
  — panel_above, 313
  — panel_below, 313
panels panel, associate application data
  — panel_userptr, 321
  — set_panel_userptr, 321
panels panel, get or set current window
  — panel_window, 322
  — replace_panel, 322
panels virtual screen refresh routine
  — panel_update, 320
  — update_panel, 320
panels window on virtual screen, move
  — move_panel, 314
  — panel_move, 314
pecho_wchar — add character and refresh window, 323
pechochar — add character and refresh window, 323
pechowchar — create and display curses pads, 142
pechowchar — create and display curses pads, 142
plot — graphics interface, 324
  Link Editor, 325
pnoutrefresh — create and display curses pads, 142
pnoutrefresh — create or refresh a pad or subpad, 304
point — graphics interface, 324
prefresh — create and display curses pads, 142
prefresh — create or refresh a pad or subpad, 304
print formatted output window —
  mvprintw, 300
print formatted output window —
  mvwprintw, 300
print formatted output window —
  printw, 300
print formatted output window —
  wprintw, 300
print formatted output in window —
  vw_printw, 352
print formatted output in window —
  vwprintw, 353
print formatted output in window —
  vw_scanw, 354
printw — print formatted output window, 300
push character back onto the input queue
  — unget_wch, 348
  — ungetch, 348
putp — apply padding information and output string, 327
putwin — read a window from, and write a window to, a file, 232

Q
qiflush — control flush of input and output on interrupt, 308

R
raw — set input mode controls, 45
read a window from, and write a window to, a file
— getwin, 232
— putwin, 232
redraw screen or portion of screen
— redrawwin, 328
— wredrawln, 328
redrawwin — redraw screen or portion of screen, 328
refresh — refresh windows and lines, 180
refresh windows and lines
— doupdate, 180
— refresh, 180
— wnoutrefresh, 180
— wrefresh, 180
remove a character
— delch, 172
— mvdelch, 172
— mvwdelch, 172
— wdelch, 172
remove a line
— deleteln, 175
— wdeleteln, 175
reserve screen line for dedicated purpose — ripoffline, 330
reset_prog_mode — save/restore terminal modes, 170
reset_shell_mode — save/restore terminal modes, 170
resetty — restore/save terminal modes, 329
restartterm — free space pointed to by terminal, 173
restore initial terminal environment
— endwin, 185
— isendwin, 185
restore/save terminal modes
— resetty, 329
— savetty, 329
retrieve a complex character (with rendition)
— in_wch, 256
— mvin_wch, 256
— mvwin_wch, 256
— win_wch, 256
retrieve a multibyte character string (without rendition) (continued)
— mvinnstr, 243
— mvinstr, 243
— mvwinnstr, 243
— mvwinstr, 243
— winnstr, 243
— winstr, 243
retrieve a single-byte character string (with rendition)
— inchnstr, 240
— inchstr, 240
— mvinchnstr, 240
— mvinchstr, 240
— mvwinchnstr, 240
— mvwinchstr, 240
— winchnstr, 240
— winchstr, 240
retrieve a wide character string (without rendition)
— innwstr, 245
— inwstr, 245
— mvinnwstr, 245
— mvinwstr, 245
— mvwinwstr, 245
— winwstr, 245
retrieve complex character string (with rendition)
— in_wchnstr, 257
— in_wchstr, 257
— mvwinchstr, 257
— mvwin_wchstr, 257
— mvwinnstr, 257
— mvwinwstr, 257
— win_wchnstr, 257
— win_wchstr, 257
return a single-byte character (with rendition)
— inch, 239
— mvinch, 239
— mvwinch, 239
— winch, 239
return character string used as key name
— key_name, 261
— keyname, 261
return current ERASE or KILL characters
 — erasechar, 186
 — erasurechar, 186
 — killchar, 186
 — killwchar, 186
return full terminal type name —
 longname, 267
return terminal baud rate — baudrate, 32
return the value of a terminfo capability
 — tigetflag, 345
 — tigetnum, 345
 — tigetstr, 345
 — tparm, 345
return the value of the environmental variable
TERM — termname, 342
ripoffline — reserve screen line for dedicated purpose, 330

S
save/restore terminal modes
 — def_prog_mode, 170
 — def_shell_mode, 170
 — reset_prog_mode, 170
 — reset_shell_mode, 170
savetty — restore/save terminal modes, 329
scanw — convert formatted input from a window, 301
scr_dump — write screen contents to/from a file, 331
scr_init — write screen contents to/from a file, 331
scr_restore — write screen contents to/from a file, 331
scr_set — write screen contents to/from a file, 331
screen initialization functions
 — initscr, 242
 — newterm, 242
scrl — scroll a window, 332
scroll — scroll a window, 332
scroll a window
 — scr, 332
 — scroll, 332
 — wscroll, 332
scrollok — set terminal output controls, 48
set or get the background character (and rendition) of window — bkgd, 34
set or get the background character (and rendition) of window — bkgdset, 34
set or get the background character (and rendition) of window — getbkgd, 34
set or get the background character (and rendition) of window — wbkgd, 34
set or get the background character (and rendition) of window — wbkgdset, 34
set a cchar_t type character from a wide character and rendition — setchar, 333
set blocking or non-blocking read —
nodelay, 307
set/clear window attributes
 — standend, 338
 — standout, 338
 — wstandend, 338
 — wstandout, 338
set_curterm — free space pointed to by terminal, 173
set input mode controls
 — cbreak, 45
 — nobreak, 45
 — noraw, 45
 — raw, 45
set or get the background character (and rendition) of window using a complex character
 — bkgnd, 36
 — bkgndset, 36
 — getbkgnd, 36
 — wbkgnd, 36
 — wbkgndset, 36
 — wgetbkgnd, 36
set_term — switch between terminals, 334
set terminal output controls
 — clearok, 48
 — idlok, 48
 — leaveok, 48
 — scrollbar, 48
 — setscrreg, 48
 — wssetscrreg, 48
set timed blocking or non-blocking read
 — notimeout, 309
 — timeout, 309
 — wtimeout, 309
set visibility of cursor — curs_set, 151
setchar — set a cchar_t type character from a wide character and rendition, 333
setscrreg — set terminal output controls, 48
setterm — free space pointed to by terminal, 173
setupterm — free space pointed to by terminal, 173
sleep process for a specified length of time — napms, 303
slk_attr_off — soft label functions, 335
slk_attr_on — soft label functions, 335
slk_attr_set — soft label functions, 335
slk_attroff — soft label functions, 335
slk_attron — soft label functions, 335
slk_attrset — soft label functions, 335
slk_clear — soft label functions, 335
slk_color — soft label functions, 335
slk_init — soft label functions, 335
slk_label — soft label functions, 335
slk_noutrefresh — soft label functions, 335
slk_refresh — soft label functions, 335
slk_restore — soft label functions, 335
slk_set — soft label functions, 335
slk_touch — soft label functions, 335
slk_wset — soft label functions, 335
soft label functions — slk_attroff, 335
soft label functions — slk_attr_on, 335
soft label functions — slk_attr_set, 335
soft label functions — slk_attroff, 335
soft label functions — slk_attr_off, 335
soft label functions — slk_attron, 335
soft label functions — slk_attr_set, 335
soft label functions — slk_attroff, 335
soft label functions — slk_attron, 335
soft label functions — slk_attr_set, 335
slk_noutrefresh — soft label functions, 335
slk_refresh — soft label functions, 335
slk_restore — soft label functions, 335
slk_set — soft label functions, 335
slk_touch — soft label functions, 335
slk_wset — soft label functions, 335
space — graphics interface, 324
specify source of screen size information — use_env, 349
standend — curses character and window attribute control routines, 66
standend — set/clear window attributes, 338
standout — curses character and window attribute control routines, 66
standout — set/clear window attributes, 338
start_color — manipulate color information, 42
stdscr — default window, 339
subpad — create and display curses pads, 142
subpad — create or refresh a pad or subpad, 304
subwin — create a new window or subwindow, 178
switch between terminals — set_term, 334
synchronize window with its parents or children
— synck, 340
— wcursyncup, 340
— wsyncdown, 340
— wsyncup, 340
synck — synchronize window with its parents or children, 340
T
term_attrs — get supported terminal video attributes, 341
termattr — get supported terminal video attributes, 341
terminfo — get supported terminal video attributes, 341
termname — return the value of the environment variable TERM, 342
tgetent — emulate the termcap database, 343
tgetflag — emulate the termcap database, 343
tgetnum — emulate the termcap database, 343
tgettext — emulate the termcap database, 343
tgoto — emulate the termcap database, 343
tigetflag — return the value of a terminfo capability, 345
tigetnum — return the value of a terminfo capability, 345
tgettext — return the value of a terminfo capability, 345
timeout — set timed blocking or non-blocking read, 309
touchline — control window refresh, 259
touchwin — control window refresh, 259
tparm — return the value of a terminfo capability, 345

tputs — apply padding information and output string, 327

typeahead — check for type-ahead characters, 346

U

unctrl — generate printable representation of a character, 347

unget_wch — push character back onto the input queue, 348

ungetch — push character back onto the input queue, 348

ungetwch — get (or push back) wchar_t characters from curses terminal keyboard, 111

untouchwin — control window refresh, 259

use complex characters (and renditions) to draw borders
  — border_set, 40
  — box_set, 40
  — wborder_set, 40

use complex characters (and renditions) to draw lines
  — hline_set, 236
  — mvhline_set, 236
  — mvwvline_set, 236
  — mvwhline_set, 236
  — mvvline_set, 236
  — vline_set, 236
  — whline_set, 236
  — wvline_set, 236

use_env — specify source of screen size information, 349

use single-byte characters (and renditions) to draw lines
  — hline, 235
  — mvhline, 235
  — mvvline, 235
  — mvwhline, 235
  — mvwvline, 235
  — vline, 235
  — whline, 235
  — wvline, 235

V

vid_attr — output attributes to the terminal, 350

vid_puts — output attributes to the terminal, 350

vidattr — output attributes to the terminal, 350

vidputs — output attributes to the terminal, 350

vline — use single-byte characters (and renditions) to draw lines, 235

vline_set — use complex characters (and renditions) to draw lines, 236

vw_printw — print formatted output in window, 352

vwprintw — print formatted output in window, 353

vw_scanw — print formatted output in window, 354

vwscanf — convert formatted input from a window, 355

W

wadd_wch — add a complex character (with rendition) to a window, 25

wadd wchar — copy a string of complex characters (with renditions) to a window, 27

wadd wchar — copy a string of complex characters (with renditions) to a window, 27

waddch — add a character (with rendition) to a window, 18

wadd ch — copy a character string (with renditions) to a window, 20

waddch — copy a character string (with renditions) to a window, 20

waddnstr — add a multi-byte character string (without rendition) to a window, 21

waddnwstr — add a wide-character string (without rendition) to a window, 21

waddnstr — add a wide-character string to a window, 23

waddstr — add a multi-byte character string (without rendition) to a window, 21
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>waddwch</td>
<td>add a wchar_t character (with attributes) to a curses window and advance cursor, 59</td>
</tr>
<tr>
<td>waddwchnstr</td>
<td>add string of wchar_t characters (and attributes) to a curses window, 62</td>
</tr>
<tr>
<td>waddwstr</td>
<td>add a string of wchar_t characters to a curses window and advance cursor, 64</td>
</tr>
<tr>
<td>waddwstr</td>
<td>add a wide-character string to a window, 23</td>
</tr>
<tr>
<td>wadjcurspos</td>
<td>moving the cursor by character, 65</td>
</tr>
<tr>
<td>wattr_get</td>
<td>control window attributes, 29</td>
</tr>
<tr>
<td>wattr_off</td>
<td>control window attributes, 29</td>
</tr>
<tr>
<td>wattr_on</td>
<td>control window attributes, 29</td>
</tr>
<tr>
<td>wattr_set</td>
<td>control window attributes, 29</td>
</tr>
<tr>
<td>wattrroff</td>
<td>change foreground window attributes, 31</td>
</tr>
<tr>
<td>wattrroff</td>
<td>curses character and window attribute control routines, 66</td>
</tr>
<tr>
<td>wattron</td>
<td>change foreground window attributes, 31</td>
</tr>
<tr>
<td>wattron</td>
<td>curses character and window attribute control routines, 66</td>
</tr>
<tr>
<td>wattrset</td>
<td>change foreground window attributes, 31</td>
</tr>
<tr>
<td>wattrset</td>
<td>curses character and window attribute control routines, 66</td>
</tr>
<tr>
<td>wbkgd</td>
<td>set or get the background character (and rendition) of window, 34</td>
</tr>
<tr>
<td>wbkgdset</td>
<td>set or get the background character (and rendition) of window, 34</td>
</tr>
<tr>
<td>wbkgnd — set or get the background character (and rendition) of window using a complex character, 36</td>
<td></td>
</tr>
<tr>
<td>wbkgndset</td>
<td>set or get the background character (and rendition) of window using a complex character, 36</td>
</tr>
<tr>
<td>wborder</td>
<td>add a single-byte border to a window, 38</td>
</tr>
<tr>
<td>wborder_set</td>
<td>use complex characters (and renditions) to draw borders, 40</td>
</tr>
<tr>
<td>wchgt</td>
<td>change the rendition of characters in a window, 46</td>
</tr>
<tr>
<td>wclear</td>
<td>clear a window, 47</td>
</tr>
<tr>
<td>wcrtobot</td>
<td>clear to the end of a window, 50</td>
</tr>
<tr>
<td>wcrtoeol</td>
<td>clear to the end of a line, 51</td>
</tr>
<tr>
<td>wcolor_set</td>
<td>control window attributes, 29</td>
</tr>
<tr>
<td>wcurlsyncup</td>
<td>synchronize window with its parents or children, 340</td>
</tr>
<tr>
<td>wdch</td>
<td>remove a character, 172</td>
</tr>
<tr>
<td>wdeleteln</td>
<td>remove a line, 175</td>
</tr>
<tr>
<td>wecho_wchar</td>
<td>add a complex character and refresh window, 184</td>
</tr>
<tr>
<td>wechochar</td>
<td>add a single-byte character and refresh window, 183</td>
</tr>
<tr>
<td>wechowchar</td>
<td>add a wchar_t character (with attributes) to a curses window and advance cursor, 59</td>
</tr>
<tr>
<td>wclear</td>
<td>clear a window, 47</td>
</tr>
<tr>
<td>wget_wch</td>
<td>get a wide character from terminal, 230</td>
</tr>
<tr>
<td>wget_wstr</td>
<td>get a wide character string from terminal, 229</td>
</tr>
<tr>
<td>wgetbkgrnd</td>
<td>set or get the background character (and rendition) of window using a complex character, 36</td>
</tr>
<tr>
<td>wgetch</td>
<td>get a single-byte character (and rendition) of window using a complex character, 36</td>
</tr>
<tr>
<td>wgetn_wstr</td>
<td>get a wide character string from terminal, 222</td>
</tr>
<tr>
<td>wgetnstr</td>
<td>get a multibyte character string from terminal, 227</td>
</tr>
<tr>
<td>wgetnwstr</td>
<td>get wchar_t character strings from curses terminal keyboard, 116</td>
</tr>
<tr>
<td>wgetstr</td>
<td>get a multibyte character string from terminal, 227</td>
</tr>
<tr>
<td>wgetwch</td>
<td>get (or push back) wchar_t characters from curses terminal keyboard, 111</td>
</tr>
<tr>
<td>wgetwstr</td>
<td>get wchar_t character strings from curses terminal keyboard, 116</td>
</tr>
<tr>
<td>whline</td>
<td>use single-byte characters (and renditions) to draw lines, 235</td>
</tr>
<tr>
<td>whline_set</td>
<td>use single-byte characters (and renditions) to draw lines, 236</td>
</tr>
<tr>
<td>win_wch</td>
<td>retrieve a complex character (with rendition), 256</td>
</tr>
<tr>
<td>win_wchnstr</td>
<td>retrieve complex character string (with rendition), 257</td>
</tr>
<tr>
<td>win_wchstr</td>
<td>retrieve complex character string (with rendition), 257</td>
</tr>
</tbody>
</table>
winch — return a single-byte character (with rendition), 239
winchnstr — retrieve a single-byte character string (with rendition), 240
winchstr — retrieve a single-byte character string (without rendition), 240
winnstr — retrieve a multibyte character string (without rendition), 243
winnwstr — get a string of wchar_t characters from a curses window, 134
winnwstr — retrieve a wide character string (without rendition), 245
wins_nwstr — insert a wide character string, 252
wins_wch — insert a complex character, 254
wins_wstr — insert a wide character string, 252
winsch — insert a character, 247
winsdelln — insert/delete lines to/from the window, 248
winsertln — insert a line in a window, 249
winsnstr — insert a multibyte character string, 250
winsnwstr — insert wchar_t string before character under the cursor in a curses window, 130
winsstr — insert a multibyte character string, 250
winstr — retrieve a multibyte character string (without rendition), 243
winsvch — insert a wchar_t character before the character under the cursor in a curses window, 129
winswstr — insert wchar_t string before character under the cursor in a curses window, 130
winwch — get a wchar_t character and its attributes from a curses window, 132
winwchnstr — get a string of wchar_t characters (and attributes) from a curses window, 133
winwchstr — get a string of wchar_t characters (and attributes) from a curses window, 133
winwstr — get a string of wchar_t characters from a curses window, 134
winwstr — retrieve a wide character string (without rendition), 245
wmove — move cursor in window, 297
wmovenextch — moving the cursor by character, 65
wmoveprevch — moving the cursor by character, 65
wnoutrefresh — refresh windows and lines, 180
wprintw — print formatted output window, 300
wredrawln — redraw screen or portion of screen, 328
wrefresh — refresh windows and lines, 180
write screen contents to/from a file
— scr_dump, 331
— scr_init, 331
— scr_restore, 331
— scr_set, 331
wscanw — convert formatted input from a window, 301
wscr — scroll a window, 332
wsetscren — set terminal output controls, 48
wstandend — curses character and window attribute control routines, 66
wstandend — set/clear window attributes, 338
wstandout — curses character and window attribute control routines, 66
wstandout — set/clear window attributes, 338
wsyncdown — synchronize window with its parents or children, 340
wsyncup — synchronize window with its parents or children, 340
wtimeout — set timed blocking or non-blocking read, 309
wtouchln — control window refresh, 259
wunctrl — generate printable representation of a wide character, 356
wvline — use single-byte characters (and renditions) to draw lines, 235
wvline_set — use complex characters (and renditions) to draw lines, 236