

NAME

ncurses – character-cell terminal interface with optimized output

SYNOPSIS

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

DESCRIPTION

The “new curses” library offers the programmer a terminal-independent means of reading keyboard and mouse input and writing to character-cell displays with output optimized to minimize screen updates. *ncurses* replaces the *curses* libraries from System V Release 4 Unix (“SVr4”) and 4.4BSD Unix, the development of which ceased in the 1990s. This document describes *ncurses* version 6.5 (patch 20251220).

ncurses permits control of the terminal screen’s contents; abstraction and subdivision thereof with *windows* and *pads*; acquisition of keyboard and mouse events; selection of color and rendering attributes (such as bold or underline); the definition and use of *soft label* keys; access to the *terminfo* terminal capability database; a *termcap* compatibility interface; and an abstraction of the system’s API for manipulating the terminal (such as *termios*(3)).

ncurses implements the interface described by X/Open Curses Issue 7. In many behavioral details not standardized by X/Open, *ncurses* emulates the *curses* library of SVr4 and provides numerous useful extensions.

ncurses man pages employ several sections to clarify matters of usage and interoperability with other *curses* implementations.

- “NOTES” describes issues and caveats of which any user of the *ncurses* API should be aware, such as limitations on the size of an underlying integral type or the availability of a preprocessor macro exclusive of a function definition (which prevents its address from being taken). This section also describes implementation details of significance to the programmer but which are not standardized.
- “EXTENSIONS” presents *ncurses* innovations beyond the X/Open Curses standard and/or the SVr4 *curses* implementation. They are termed *extensions* to indicate that they cannot be implemented solely by using the library API, but require access to the library’s internal state.
- “PORTABILITY” discusses matters (beyond the exercise of extensions) that should be considered when writing to a *curses* standard, or for multiple implementations.
- “HISTORY” examines points of detail in *ncurses* and other *curses* implementations over the decades of their development, particularly where precedent or inertia have frustrated better design (and, in a few cases, where such inertia has been overcome).

A *curses* application must be linked with the library; use the **-lncurses** option to your compiler or linker. A debugging version of the library may be available; if so, link with it using **-lncurses_g**. (Your system integrator may have installed these libraries such that you can use the options **-lncurses** and **-lncurses_g**, respectively.) Then *ncurses_g* library logs events describing *ncurses* actions to a file called *trace* in the application’s working directory at startup. See section “ALTERNATE CONFIGURATIONS” below.

Application Structure

A *curses* application uses information from the system locale; *setlocale*(3) prepares it for *curses* library calls.

```
setlocale(LC_ALL, "");
```

If the locale is not thus initialized, the library assumes that characters are printable as in ISO 8859-1, to work with certain legacy programs. You should initialize the locale; do not expect consistent behavior from the library when the locale has not been set up.

initscr(3X) or **newterm**(3X) must be called to initialize *curses* before use of any functions that access or manipulate windows or screens.

To get character-at-a-time input without echoing — most interactive, screen-oriented programs want this — use the following sequence.

```
initscr(); cbreak(); noecho();
```

Most applications would perform further setup as follows.

```
noqiflush();
keypad(stdscr, TRUE);
```

A *curses* program then often enters an event-handling loop. Call **endwin**(3X) before exiting.

Overview

A *curses* library abstracts the terminal with a *SCREEN* data structure, and represents all or part of its display with *WINDOW* structures. Distinct properties apply to each; for example, the *line discipline* of a typical Unix terminal driver is in one of three modes: raw, cbreak, or canonical (“cooked”). In *ncurses*, the line discipline is a property of the screen, applying identically to all windows associated with it.

A *window* is a rectangular grid of character cells, addressed by line and column coordinates (*y*, *x*), with the upper left corner as (0, 0). A window called **stdscr**, by default the same size as the terminal screen, is always available. Create others with **newwin**(3X).

A *curses* library does not manage overlapping windows (but see below). You can either use **stdscr** to manage one screen-filling window, or tile the screen into non-overlapping windows and not use **stdscr** at all. Mixing the two approaches will result in unpredictable and undesired effects.

Functions permit manipulation of a window and the *cursor* identifying the cell within it at which the next operation will occur. Among those, the most basic are **move**(3X) and **addch**(3X): these place the cursor within and write a character to **stdscr**, respectively.

Frequent changes to the terminal screen can cause unpleasant flicker or inefficient use of the communication channel to the device, so as a rule the library does not update it automatically. Therefore, after using *curses* functions to accumulate a set of desired updates that make sense to present together, call **refresh**(3X) to tell the library to make the user’s screen look like **stdscr**. The library *optimizes* its output by computing a minimal volume of operations to mutate the screen from its state at the previous refresh to the new one. Effective optimization demands accurate information about the terminal device: the management of such information is the province of the **terminfo**(3X) API, a feature of every standard *curses* implementation.

Special windows called *pads* may also be manipulated. These are not constrained to the size of the terminal screen and their contents need not be completely displayed. See **curs_pad**(3X).

Many terminals support configuration of character cell foreground and background colors as well as *attributes*, which cause characters to render in such modes as boldfaced, underlined, or in reverse video. See **curs_color**(3X) and **curs_attr**(3X).

curses defines constants to simplify access to a small set of forms-drawing graphics corresponding to the DEC Alternate Character Set (ACS), a feature of VT100 and other terminals. See **addch**(3X).

curses is implemented using the operating system’s terminal driver; key events are received not as scan codes but as byte sequences. The driver reports graphical keycaps (alphanumeric and punctuation keys, and the space) as-is. Everything else, including the tab, enter/return, keypad, arrow, and function keys, appears to *curses* as a control character or a multibyte *escape sequence*. *curses* can translate the latter into unique *key codes*. See **keypad**(3X) and **getch**(3X).

ncurses provides reimplementations of the SVr4 **panel**(3X), **form**(3X), and **menu**(3X) libraries; they permit overlapping windows and ease construction of user interfaces with *curses*.

Initialization

The selection of an appropriate value of *TERM* in the process environment is essential to correct *curses* and *terminfo* library operation. A well-configured system selects a correct *TERM* value automatically; **tset**(1) may assist with troubleshooting exotic situations.

If you change the terminal type from a shell, export *TERM*, then run **tset**(1) or the “**tput init**” command. See subsection “Tabs and Initialization” of **terminfo**(5).

If the environment variables *LINES* and *COLUMNS* are set, or if the *curses* program is executing in a graphical windowing environment, the information obtained thence overrides that obtained by *terminfo*. An *ncurses* extension supports resizable terminal displays; see **wresize**(3X).

If the environment variable *TERMINFO* is defined, a *curses* program checks first for a terminal type description in the location it identifies. *TERMINFO* is useful for developing type descriptions or when write permission to */usr/share/terminfo* is not available.

See section “ENVIRONMENT” below.

Naming Conventions

curses offers many functions in variant forms using a regular set of alternatives to the name of an elemental one. Those prefixed with “w” require a *WINDOW* pointer argument; those with a “mv” prefix first perform cursor movement using **wmove**(3X); a “mvw” prefix indicates both. The “w” function is typically the elemental one; the removal of this prefix usually indicates operation on **stdscr**.

Four functions prefixed with “p” require a *pad* argument; see below.

In function synopses, *ncurses* man pages apply the following names to parameters. We introduce the character types in the next subsection.

<i>bf</i>	a <i>bool</i> (TRUE or FALSE)
<i>c</i>	a <i>char</i> or <i>int</i>
<i>ch</i>	a <i>chtype</i>
<i>wc</i>	a <i>wchar_t</i> or <i>wint_t</i>
<i>wch</i>	a <i>cchar_t</i>
<i>win</i>	pointer to a <i>WINDOW</i>
<i>pad</i>	pointer to a <i>WINDOW</i> that is a pad
<i>pair</i>	a foreground/background color pair identifier

Wide and Non-wide Character Configurations

This man page primarily surveys functions that appear in any configuration of the library. There are two common configurations; for others, see section “ALTERNATE CONFIGURATIONS” below.

ncurses is the library in its “non-wide” configuration, handling only eight-bit characters. It stores a character combined with attributes and a color pair identifier in a *chtype* datum, which is often an alias of *int*. A string of *ncurses* characters is similar to a C *char* string; a *chtype* string ends with an integral **0**, the null *curses* character.

Attributes and a color pair identifier (with no corresponding character) can be stored in variables of *chtype* or *attr_t* type. In either case, they are accessed via an integral bit mask.

ncurses stores each cell of a *WINDOW* as a *chtype*. X/Open Curses does not specify the sizes of the character code or color pair identifier, nor the quantity of attribute bits, in *chtype*; these are implementation-dependent. *ncurses* uses eight bits for the character code. An application requiring a wider character type, for instance to represent Unicode, should use the wide-character API.

ncursesw is the library in its “wide” configuration, which handles character encodings requiring a larger data type than *char* (a byte-sized type) can represent. It provides additional functions that complement those in the non-wide library where the size of the underlying character type is significant. A somewhat regular naming convention relates many of the wide variants to their non-wide counterparts; where a non-wide function name contains “ch” or “str”, prefix it with “_w” to obtain the wide counterpart. For example, **waddch** becomes **wadd_wch**. An exception is **ins_nwstr** (and its variants), spelled thus instead of “insn_wstr”. (Exceptions that add only “w” comprise **addwstr**, **inwstr**, and their variants.)

This convention is inapplicable to some non-wide function names, so other transformations are used for the wide configuration: the window background management function “bkgd” becomes “bkgrnd”; the window border-drawing and -clearing functions are suffixed with “_set”; and character attribute manipulation functions like “attron” become “attr_on”.

cchar_t is a *curses complex character* and corresponds to the non-wide-character configuration’s *chtype*. It is a structure type because it requires more storage than a standard scalar type offers. A character code may not be representable as a *char*, and moreover more than one character may occupy a cell (as with accent marks and other

diacritics). Each character is of type `wchar_t`; a complex character contains one spacing character and zero or more non-spacing characters (see below). A string of complex characters ends with a `wchar_t` whose `wchar_t` member is the null wide character. Attributes and a color pair identifier are stored in separate fields of the structure, not combined into an integer as in `chtype`.

`ncurses` stores each cell of a `WINDOW` as a `cchar_t`. **setcchar(3X)** and **getcchar(3X)** store and retrieve `cchar_t` data.

The wide library API of `ncurses` depends on two data types standardized by ISO C95.

`wchar_t` stores a wide character. Like `chtype`, it may be an alias of `int`. Depending on the character encoding, a wide character may be *spacing*, meaning that it occupies a character cell by itself and typically accompanies cursor advancement, or *non-spacing*, meaning that it occupies the same cell as a spacing character, is often regarded as a “modifier” of the base glyph with which it combines, and typically does not advance the cursor.

`wint_t` can store a `wchar_t` or the constant `WEOF`, analogously to the `int`-sized character manipulation functions of ISO C and its constant `EOF`.

Function Name Index

The following table lists the `curses` functions provided in the non-wide and wide APIs and the corresponding man pages that describe them. Those flagged with “*” are `ncurses`-specific, neither described by X/Open Curses nor present in SVr4.

<i>curses</i> Function Name	Man Page
COLOR_PAIR	curl_color(3X)
PAIR_NUMBER	curl_color(3X)
add_wch	curl_add_wch(3X)
add_wchnstr	curl_add_wchstr(3X)
add_wchstr	curl_add_wchstr(3X)
addch	curl_addch(3X)
addchnstr	curl_addchstr(3X)
addchstr	curl_addchstr(3X)
addnstr	curl_addstr(3X)
addnwstr	curl_addwstr(3X)
addstr	curl_addstr(3X)
addwstr	curl_addwstr(3X)
alloc_pair	new_pair(3X)*
assume_default_colors	default_colors(3X)*
attr_get	curl_attr(3X)
attr_off	curl_attr(3X)
attr_on	curl_attr(3X)
attr_set	curl_attr(3X)
attroff	curl_attr(3X)
attron	curl_attr(3X)
attrset	curl_attr(3X)
baudrate	curl_termattrs(3X)
beep	curl_beep(3X)
bkgd	curl_bkgd(3X)
bkgdset	curl_bkgd(3X)
bkgrnd	curl_bkgrnd(3X)
bkgrndset	curl_bkgrnd(3X)
border	curl_border(3X)
border_set	curl_border_set(3X)
box	curl_border(3X)

box_set	curls_border_set(3X)
can_change_color	curls_color(3X)
cbreak	curls_inopts(3X)
chgat	curls_attr(3X)
clear	curls_clear(3X)
clearok	curls_outopts(3X)
clrtoebot	curls_clear(3X)
clrtoeol	curls_clear(3X)
color_content	curls_color(3X)
color_set	curls_attr(3X)
copywin	curls_overlay(3X)
curls_set	curls_kernel(3X)
curses_trace	curls_trace(3X)*
curses_version	curls_extend(3X)*
def_prog_mode	curls_kernel(3X)
def_shell_mode	curls_kernel(3X)
define_key	define_key(3X)*
del_curlterm	curls_terminfo(3X)
delay_output	curls_util(3X)
delch	curls_delch(3X)
deleteln	curls_deleteln(3X)
delscreen	curls_initscr(3X)
delwin	curls_window(3X)
derwin	curls_window(3X)
doupdate	curls_refresh(3X)
dupwin	curls_window(3X)
echo	curls_inopts(3X)
echo_wchar	curls_add_wch(3X)
echochar	curls_addch(3X)
endwin	curls_initscr(3X)
erase	curls_clear(3X)
erasechar	curls_termattrs(3X)
erasewchar	curls_termattrs(3X)
exit_curses	curls_memleaks(3X)*
exit_terminfo	curls_memleaks(3X)*
extended_color_content	curls_color(3X)*
extended_pair_content	curls_color(3X)*
extended_slk_color	curls_slk(3X)*
filter	curls_util(3X)
find_pair	new_pair(3X)*
flash	curls_beep(3X)
flushinp	curls_util(3X)
free_pair	new_pair(3X)*
get_escdelay	curls_threads(3X)*
get_wch	curls_get_wch(3X)
get_wstr	curls_get_wstr(3X)
getattro	curls_attr(3X)
getbegx	curls_legacy(3X)*
getbegy	curls_legacy(3X)*
getbegyx	curls_getyx(3X)
getbkgd	curls_bkgd(3X)
getbkgrnd	curls_bkgrnd(3X)
getcchar	curls_getcchar(3X)

getch	curs_getch(3X)
getcurx	curs_legacy(3X)*
getcury	curs_legacy(3X)*
getmaxx	curs_legacy(3X)*
getmaxy	curs_legacy(3X)*
getmaxyx	curs_getyx(3X)
getmouse	curs_mouse(3X)*
getn_wstr	curs_get_wstr(3X)
getnstr	curs_getstr(3X)
getparx	curs_legacy(3X)*
getpary	curs_legacy(3X)*
getparyx	curs_getyx(3X)
getstr	curs_getstr(3X)
getsyx	curs_kernel(3X)
getwin	curs_util(3X)
getyx	curs_getyx(3X)
halfdelay	curs_inopts(3X)
has_colors	curs_color(3X)
has_ic	curs_termattrs(3X)
has_il	curs_termattrs(3X)
has_key	curs_getch(3X)*
has_mouse	curs_mouse(3X)*
hline	curs_border(3X)
hline_set	curs_border_set(3X)
idcok	curs_outopts(3X)
idlok	curs_outopts(3X)
immedok	curs_outopts(3X)
in_wch	curs_in_wch(3X)
in_wchnstr	curs_in_wchstr(3X)
in_wchstr	curs_in_wchstr(3X)
inch	curs_inch(3X)
inchnstr	curs_inchstr(3X)
inchstr	curs_inchstr(3X)
init_color	curs_color(3X)
init_extended_color	curs_color(3X)*
init_extended_pair	curs_color(3X)*
init_pair	curs_color(3X)
initscr	curs_initscr(3X)
innstr	curs_instr(3X)
innwstr	curs_inwstr(3X)
ins_nwstr	curs_ins_wstr(3X)
ins_wch	curs_ins_wch(3X)
ins_wstr	curs_ins_wstr(3X)
insch	curs_insch(3X)
insdelln	curs_deleteln(3X)
insertln	curs_deleteln(3X)
insnstr	curs_insstr(3X)
insstr	curs_insstr(3X)
instr	curs_instr(3X)
intrflush	curs_inopts(3X)
inwstr	curs_inwstr(3X)
is_cbreak	curs_inopts(3X)*
is_cleared	curs_opaque(3X)*

is_echo	curl_inopts(3X)*
is_idcok	curl_opaque(3X)*
is_idlok	curl_opaque(3X)*
is_immedok	curl_opaque(3X)*
is_keypad	curl_opaque(3X)*
is_leaveok	curl_opaque(3X)*
is_linetouched	curl_touch(3X)
is_nl	curl_inopts(3X)*
is_nodelay	curl_opaque(3X)*
is_notimeout	curl_opaque(3X)*
is_pad	curl_opaque(3X)*
is_raw	curl_inopts(3X)*
is_scrollok	curl_opaque(3X)*
is_subwin	curl_opaque(3X)*
is_syncok	curl_opaque(3X)*
is_term_resized	resizeterm(3X)*
is_wintouched	curl_touch(3X)
isendwin	curl_initscr(3X)
key_defined	key_defined(3X)*
key_name	curl_util(3X)
keybound	keybound(3X)*
keyname	curl_util(3X)
keyok	keyok(3X)*
keypad	curl_inopts(3X)
killchar	curl_termattrs(3X)
killwchar	curl_termattrs(3X)
leaveok	curl_outopts(3X)
longname	curl_termattrs(3X)
mcprint	curl_print(3X)*
meta	curl_inopts(3X)
mouse_trafo	curl_mouse(3X)*
mouseinterval	curl_mouse(3X)*
mousemask	curl_mouse(3X)*
move	curl_move(3X)
mvadd_wch	curl_add_wch(3X)
mvadd_wchnstr	curl_add_wchstr(3X)
mvadd_wchstr	curl_add_wchstr(3X)
mvaddch	curl_addch(3X)
mvaddchnstr	curl_addchstr(3X)
mvaddchstr	curl_addchstr(3X)
mvaddnstr	curl_addstr(3X)
mvaddnwstr	curl_addwstr(3X)
mvaddstr	curl_addstr(3X)
mvaddwstr	curl_addwstr(3X)
mvchgat	curl_attr(3X)
mvcur	curl_kernel(3X)
mvdelch	curl_delch(3X)
mvderwin	curl_window(3X)
mvget_wch	curl_get_wch(3X)
mvget_wstr	curl_get_wstr(3X)
mvgetch	curl_getch(3X)
mvgetn_wstr	curl_get_wstr(3X)
mvgetnstr	curl_getstr(3X)

mvgetstr	curls_getstr(3X)
mvhline	curls_border(3X)
mvhline_set	curls_border_set(3X)
mvin_wch	curls_in_wch(3X)
mvin_wchnstr	curls_in_wchstr(3X)
mvin_wchstr	curls_in_wchstr(3X)
mvinch	curls_inch(3X)
mvinchnstr	curls_inchstr(3X)
mvinchstr	curls_inchstr(3X)
mvinnstr	curls_instr(3X)
mvinnwstr	curls_inwstr(3X)
mvins_nwstr	curls_ins_wstr(3X)
mvins_wch	curls_ins_wch(3X)
mvins_wstr	curls_ins_wstr(3X)
mvinsch	curls_insch(3X)
mvinsnstr	curls_insstr(3X)
mvinsstr	curls_insstr(3X)
mvinstr	curls_instr(3X)
mvinwstr	curls_inwstr(3X)
mvprintw	curls_printw(3X)
mvscanw	curls_scanw(3X)
mvvline	curls_border(3X)
mvvline_set	curls_border_set(3X)
mvwadd_wch	curls_add_wch(3X)
mvwadd_wchnstr	curls_add_wchstr(3X)
mvwadd_wchstr	curls_add_wchstr(3X)
mvwaddch	curls_addch(3X)
mvwaddchnstr	curls_addchstr(3X)
mvwaddchstr	curls_addchstr(3X)
mvwaddnstr	curls_addstr(3X)
mvwaddnwstr	curls_addwstr(3X)
mvwaddstr	curls_addstr(3X)
mvwaddwstr	curls_addwstr(3X)
mvwchgat	curls_attr(3X)
mvwdelch	curls_delch(3X)
mvwget_wch	curls_get_wch(3X)
mvwget_wstr	curls_get_wstr(3X)
mvwgetch	curls_getch(3X)
mvwgetn_wstr	curls_get_wstr(3X)
mvwgetnstr	curls_getstr(3X)
mvwgetstr	curls_getstr(3X)
mvwhline	curls_border(3X)
mvwhline_set	curls_border_set(3X)
mvwin	curls_window(3X)
mvwin_wch	curls_in_wch(3X)
mvwin_wchnstr	curls_in_wchstr(3X)
mvwin_wchstr	curls_in_wchstr(3X)
mvwinch	curls_inch(3X)
mvwinchnstr	curls_inchstr(3X)
mvwinchstr	curls_inchstr(3X)
mvwinnstr	curls_instr(3X)
mvwinnwstr	curls_inwstr(3X)
mvwins_nwstr	curls_ins_wstr(3X)

mvwins_wch	curls_ins_wch(3X)
mvwins_wstr	curls_ins_wstr(3X)
mvwinsch	curls_insch(3X)
mvwinsnstr	curls_insstr(3X)
mvwinsstr	curls_insstr(3X)
mvwinstr	curls_instr(3X)
mvwinwstr	curls_inwstr(3X)
mvwprintw	curls_printw(3X)
mvwscanw	curls_scanw(3X)
mvwvline	curls_border(3X)
mvwvline_set	curls_border_set(3X)
napms	curls_kernel(3X)
newpad	curls_pad(3X)
newterm	curls_initscr(3X)
newwin	curls_window(3X)
nl	curls_inopts(3X)
nocbreak	curls_inopts(3X)
nodelay	curls_inopts(3X)
noecho	curls_inopts(3X)
nofilter	curls_util(3X)*
nonl	curls_inopts(3X)
noqiflush	curls_inopts(3X)
noraw	curls_inopts(3X)
notimeout	curls_inopts(3X)
overlay	curls_overlay(3X)
overwrite	curls_overlay(3X)
pair_content	curls_color(3X)
pecho_wchar	curls_pad(3X)
pechochar	curls_pad(3X)
pnoutrefresh	curls_pad(3X)
prefresh	curls_pad(3X)
printw	curls_printw(3X)
putp	curls_terminfo(3X)
putwin	curls_util(3X)
qiflush	curls_inopts(3X)
raw	curls_inopts(3X)
redrawwin	curls_refresh(3X)
refresh	curls_refresh(3X)
reset_color_pairs	curls_color(3X)*
reset_prog_mode	curls_kernel(3X)
reset_shell_mode	curls_kernel(3X)
resetty	curls_kernel(3X)
resize_term	resizeterm(3X)*
resizeterm	resizeterm(3X)*
restartterm	curls_terminfo(3X)
ripline	curls_kernel(3X)
savetty	curls_kernel(3X)
scanw	curls_scanw(3X)
scr_dump	curls_scr_dump(3X)
scr_init	curls_scr_dump(3X)
scr_restore	curls_scr_dump(3X)
scr_set	curls_scr_dump(3X)
scrll	curls_scroll(3X)

scroll	curl_scroll(3X)
scrollok	curl_outopts(3X)
set_curterm	curl_terminfo(3X)
set_escdelay	curl_threads(3X)*
set_tabsize	curl_threads(3X)*
set_term	curl_initscr(3X)
setcchar	curl_getcchar(3X)
setscreg	curl_outopts(3X)
setsyx	curl_kernel(3X)
setupterm	curl_terminfo(3X)
slk_attr	curl_slk(3X)*
slk_attr_off	curl_slk(3X)
slk_attr_on	curl_slk(3X)
slk_attr_set	curl_slk(3X)
slk_attroff	curl_slk(3X)
slk_atron	curl_slk(3X)
slk_attrset	curl_slk(3X)
slk_clear	curl_slk(3X)
slk_color	curl_slk(3X)
slk_init	curl_slk(3X)
slk_label	curl_slk(3X)
slk_noutrefresh	curl_slk(3X)
slk_refresh	curl_slk(3X)
slk_restore	curl_slk(3X)
slk_set	curl_slk(3X)
slk_touch	curl_slk(3X)
slk_wset	curl_slk(3X)
standend	curl_attr(3X)
standout	curl_attr(3X)
start_color	curl_color(3X)
subpad	curl_pad(3X)
subwin	curl_window(3X)
syncok	curl_window(3X)
term_attrs	curl_termattrs(3X)
termattrs	curl_termattrs(3X)
termname	curl_termattrs(3X)
tgetent	curl_termcap(3X)
tgetflag	curl_termcap(3X)
tgetnum	curl_termcap(3X)
tgetstr	curl_termcap(3X)
tgoto	curl_termcap(3X)
tigetflag	curl_terminfo(3X)
tigetnum	curl_terminfo(3X)
tigetstr	curl_terminfo(3X)
timeout	curl_inopts(3X)
tiparm	curl_terminfo(3X)
tiparm_s	curl_terminfo(3X)*
tiscan_s	curl_terminfo(3X)*
touchline	curl_touch(3X)
touchwin	curl_touch(3X)
tparm	curl_terminfo(3X)
tputs	curl_termcap(3X)
tputs	curl_terminfo(3X)

trace	curl_trace(3X)*
typeahead	curl_inopts(3X)
unctrl	curl_util(3X)
unget_wch	curl_get_wch(3X)
ungetch	curl_getch(3X)
ungetmouse	curl_mouse(3X)*
untouchwin	curl_touch(3X)
use_default_colors	default_colors(3X)*
use_env	curl_util(3X)
use_extended_names	curl_extend(3X)*
use_legacy_coding	legacy_coding(3X)*
use_screen	curl_threads(3X)*
use_tioctl	curl_util(3X)*
use_window	curl_threads(3X)*
vid_attr	curl_terminfo(3X)
vid_puts	curl_terminfo(3X)
vidattr	curl_terminfo(3X)
vidputs	curl_terminfo(3X)
vline	curl_border(3X)
vline_set	curl_border_set(3X)
vw_printw	curl_printw(3X)
vw_scanw	curl_scanw(3X)
vwprintw	curl_printw(3X)
vwscanw	curl_scanw(3X)
wadd_wch	curl_add_wch(3X)
wadd_wchnstr	curl_add_wchstr(3X)
wadd_wchstr	curl_add_wchstr(3X)
waddch	curl_addch(3X)
waddchnstr	curl_addchstr(3X)
waddchstr	curl_addchstr(3X)
waddnstr	curl_addstr(3X)
waddnwstr	curl_addwstr(3X)
waddstr	curl_addstr(3X)
waddwstr	curl_addwstr(3X)
wattr_get	curl_attr(3X)
wattr_off	curl_attr(3X)
wattr_on	curl_attr(3X)
wattr_set	curl_attr(3X)
wattroff	curl_attr(3X)
wattron	curl_attr(3X)
wattrset	curl_attr(3X)
wbkgd	curl_bkgd(3X)
wbkgdset	curl_bkgd(3X)
wbkgrnd	curl_bkgrnd(3X)
wbkgrndset	curl_bkgrnd(3X)
wborder	curl_border(3X)
wborder_set	curl_border_set(3X)
wchgat	curl_attr(3X)
wclear	curl_clear(3X)
wclrtoebot	curl_clear(3X)
wclrtoeol	curl_clear(3X)
wcolor_set	curl_attr(3X)
wcursyncup	curl_window(3X)

wdelch	 curs_delch(3X)
wdeleteln	 curs_deleteln(3X)
wecho_wchar	 curs_add_wch(3X)
wechochar	 curs_addch(3X)
wenclose	 curs_mouse(3X)*
werase	 curs_clear(3X)
wget_wch	 curs_get_wch(3X)
wget_wstr	 curs_get_wstr(3X)
wgetbkgrnd	 curs_bkgrnd(3X)
wgetch	 curs_getch(3X)
wgetdelay	 curs_opaque(3X)*
wgetn_wstr	 curs_get_wstr(3X)
wgetnstr	 curs_getstr(3X)
wgetparent	 curs_opaque(3X)*
wgetscrreg	 curs_opaque(3X)*
wgetstr	 curs_getstr(3X)
whline	 curs_border(3X)
whline_set	 curs_border_set(3X)
win_wch	 curs_in_wch(3X)
win_wchnstr	 curs_in_wchstr(3X)
win_wchstr	 curs_in_wchstr(3X)
winch	 curs_inch(3X)
winchnstr	 curs_inchstr(3X)
winchstr	 curs_inchstr(3X)
winnstr	 curs_instr(3X)
winnwstr	 curs_inwstr(3X)
wins_nwstr	 curs_ins_wstr(3X)
wins_wch	 curs_ins_wch(3X)
wins_wstr	 curs_ins_wstr(3X)
winsch	 curs_insch(3X)
winsdelln	 curs_deleteln(3X)
winsertln	 curs_deleteln(3X)
winsnstr	 curs_insstr(3X)
winsstr	 curs_insstr(3X)
winstr	 curs_instr(3X)
winwstr	 curs_inwstr(3X)
wmouse_trafo	 curs_mouse(3X)*
wmove	 curs_move(3X)
wnoutrefresh	 curs_refresh(3X)
wprintw	 curs_printw(3X)
wredrawln	 curs_refresh(3X)
wrefresh	 curs_refresh(3X)
wresize	 wresize(3X)*
wscanw	 curs_scanw(3X)
wscrl	 curs_scroll(3X)
wsetscreg	 curs_outopts(3X)
wstandend	 curs_attr(3X)
wstandout	 curs_attr(3X)
wsyncdown	 curs_window(3X)
wsyncup	 curs_window(3X)
wtimeout	 curs_inopts(3X)
wtouchln	 curs_touch(3X)
wunctrl	 curs_util(3X)

wvline	curls_border(3X)
wvline_set	curls_border_set(3X)

ncurses's screen-pointer extension adds additional functions corresponding to many of the above, each with an “_sp” suffix; see **curls_sp_funcs(3X)**.

The availability of some extensions is configurable when *ncurses* is compiled; see sections “ALTERNATE CONFIGURATIONS” and “EXTENSIONS” below.

RETURN VALUE

Unless otherwise noted, functions that return integers return the constants **OK** on success and **ERR** on failure; see **curls_variables(3X)**. Functions that return pointers return a null pointer on failure. Typically, *ncurses* treats a null pointer passed as a function parameter as a failure. Functions prefixed with “mv” first perform cursor movement and fail if the position (y, x) is outside the window boundaries.

ENVIRONMENT

The following symbols from the process environment customize the runtime behavior of *ncurses* applications. The library may be configured to disregard the variables *TERMINFO*, *TERMINFO_DIRS*, *TERMPATH*, and *HOME*, if the user is the superuser (root), or the application uses *setuid(2)* or *setgid(2)*.

BAUDRATE

The debugging library checks this variable when the application has redirected output to a file. *ncurses* interprets its integral value as the terminal's line speed in bits per second. If that value is absent or invalid, *ncurses* uses 9600. This feature allows developers to construct repeatable test cases that take into account optimization decisions that depend on the terminal's line speed.

CC (command character)

When set, the **command_character** (**cmdch**) capability value of loaded *terminfo* entries changes to the value of this variable. Very few *terminfo* entries provide this feature.

Because this name is also used in development environments to store the C compiler's name, *ncurses* ignores its value if it is not one character in length.

COLUMNS

This variable specifies the width of the screen in character cells. Applications running in a windowing environment usually are able to obtain the width of the window in which they are executing. *ncurses* enforces an upper limit of 512 when reading the value. If *COLUMNS* is not defined and the terminal's screen size is not available from the terminal driver, *ncurses* uses the size specified by the **columns** (**cols**) capability of the terminal type's entry in the *terminfo* database, if any.

It is important that your application use the correct screen size. Automatic detection thereof is not always possible because an application may be running on a host that does not honor NAWS (Negotiations About Window Size) or as a different user ID than the owner of the terminal device file. Setting *COLUMNS* and/or *LINES* overrides the library's use of the screen size obtained from the operating system.

The *COLUMNS* and *LINES* variables may be specified independently. This property is useful to circumvent misfeatures of legacy terminal type descriptions; *xterm(1)* descriptions specifying 65 lines were once notorious. For best results, avoid specifying **cols** and **lines** capability codes in *terminfo* descriptions of terminal emulators.

use_env(3X) can disable use of the process environment in determining the screen size. **use_tioctl(3X)** can update *COLUMNS* and *LINES* to match the screen size obtained from system calls or the terminal database.

ESCDELAY

For *curses* to distinguish the ESC character resulting from a user's press of the “Escape” key on the input device from one beginning an *escape sequence* (as commonly produced by function keys), it waits after receiving the escape character to see if further characters are available on the input stream within a short interval. A global variable **ESCDELAY** stores this interval in milliseconds. The default value of 1000 (one second) is adequate for most uses. This environment variable overrides it; *ncurses* enforces an upper limit of 30,000 (30 seconds) when reading the value.

The most common instance where you may wish to change this value is to work with a remote host over a

slow communication channel. If the host running a *curses* application does not receive the characters of an escape sequence in a timely manner, the library can interpret them as multiple key stroke events. Conversely, a fast typist on a low-latency connection who happens to input an ESC followed by characters that match an escape sequence may experience confusing application behavior.

xterm(1) mouse events are a form of escape sequence; therefore, if your application makes heavy use of multiple-clicking, you may wish to lengthen the default value because the delay applies to the composite multi-click event as well as the individual clicks.

Portable applications should not rely upon the presence of **ESCDELAY** in either form, but setting the environment variable rather than the global variable does not create problems when compiling an application.

If **keypad**(3X) is disabled for the *curses* window receiving input, **ESCDELAY** is irrelevant and a program must disambiguate escape sequences itself.

HOME

ncurses may read and write auxiliary terminal descriptions in *.termcap* and *.terminfo* files in the user's home directory.

LINES

This counterpart to *COLUMNS* specifies the height of the screen in characters. The corresponding *terminfo* capability and code is **lines**. See the description of the *COLUMNS* variable above.

MOUSE_BUTTONS_123

(OS/2 EMX port only) OS/2 numbers a three-button mouse inconsistently with other platforms, such that 1 is the left button, 2 the right, and 3 the middle. This variable customizes the mouse button numbering. Its value must be three digits 1–3 in any order. By default, *ncurses* assumes a numbering of "132".

NCURSES_ASSUMED_COLORS

If set, this variable overrides the *ncurses* library's compiled-in assumption that the terminal's default colors are white on black; see **default_colors**(3X). Set the foreground and background color values with this environment variable by assigning it two integer values separated by a comma, indicating foreground and background color numbers, respectively.

For example, to tell *ncurses* not to assume anything about the colors, use a value of "-1,-1". To make the default color scheme green on black on a terminal that uses ANSI X3.64/ECMA-48/ISO 6429 color assignments, use "2,0". *ncurses* accepts integral values from -1 up to the value of the *terminfo* **max_colors** (**colors**) capability for the selected terminal type.

NCURSES_CONSOLE2

(MinGW port only) The *Console2* program defectively handles the Microsoft Console API call *CreateConsoleScreenBuffer*. Applications that use it will hang. However, it is possible to simulate the action of this call by mapping coordinates, explicitly saving and restoring the original screen contents. Setting the environment variable *NCGDB* has the same effect.

NCURSES_GPM_TERMS

(Linux only) When *ncurses* is configured to use the GPM interface, this variable may list one or more terminal type names, delimited by vertical bars (|) or colons (:), against which the *TERM* variable (see below) is matched. An empty value disables the GPM interface, using *ncurses*'s built-in support for *xterm*(1) mouse protocols instead. If the variable is absent, *ncurses* attempts to open GPM if *TERM* contains "linux".

NCURSES_NO_HARD_TABS

ncurses may use tab characters in cursor movement optimization. In some cases, your terminal driver may not handle them properly. Set this environment variable to any value to disable the feature. You can also adjust your *stty*(1) settings to avoid the problem.

NCURSES_NO_MAGIC_COOKIE

Many terminals store video attributes as properties of a character cell, as *curses* does. Historically, some recorded changes in video attributes as data that logically (but invisibly) *occupied* character cells on the display, switching attributes on or off, similarly to tags in a markup language, which then had to be overprinted to depict the cells' desired contents; these are termed "magic cookies". If the *terminfo* entry for

your terminal type does not adequately describe its handling of magic cookies, set this variable to any value to instruct *ncurses* to disable attributes entirely.

NCURSES_NO_PADDING

Most terminal type descriptions in the *terminfo* database detail hardware devices. Many people use *curses*-based applications in terminal emulator programs that run in a windowing environment. These programs can duplicate all of the important features of a hardware terminal, but often lack their limitations. Chief among these absent drawbacks is the problem of data flow management; that is, limiting the speed of communication to what the hardware could handle. Unless a hardware terminal is interfaced into a terminal concentrator (which does flow control), an application must manage flow itself to prevent overruns and data loss.

A solution that comes at no hardware cost is for an application to pause transmission after directing a terminal to execute an operation that it performs slowly, such as clearing the display. Many terminal type descriptions, including that for the VT100, embed delay specifications in capabilities. You may wish to use these terminal descriptions without paying the performance penalty. Set *NCURSES_NO_PADDING* to any value to disable all but mandatory padding. Mandatory padding is used by such terminal capabilities as **flash_screen** (**flash**).

NCURSES_NO_SETBUF

(Obsolete) Prior to internal changes developed in *ncurses* 5.9 (patches 20120825 through 20130126), the library used *setbuf*(3) to enable fully buffered output when initializing the terminal. This was done, as in SVr4 *curses*, to increase performance. For testing purposes, both of *ncurses* and of certain applications, this feature was made optional. Setting this variable disabled output buffering, leaving the output stream in the original (usually line-buffered) mode.

Nowadays, *ncurses* performs its own buffering and does not require this workaround; it does not modify the buffering of the standard output stream. This approach makes the library's handling of keyboard-initiated signals more robust. A drawback is that certain unconventional programs mixed *stdio*(3) calls with *ncurses* calls and (usually) got the behavior they expected. This is no longer the case; *ncurses* does not write to the standard output file descriptor through a *stdio*-buffered stream.

As a special case, low-level API calls such as **putp**(3X) still use the standard output stream. High-level *curses* calls such as **printw**(3X) do not.

NCURSES_NO_UTF8_ACS

At initialization, *ncurses* inspects the *TERM* environment variable for special cases where VT100 forms-drawing characters (and the corresponding alternate character set *terminfo* capabilities) are known to be unsupported by terminal types that otherwise claim VT100 compatibility. Specifically, when running in a UTF-8 locale, the Linux virtual console device and the GNU *screen*(1) program ignore them. Set this variable to a nonzero value to instruct *ncurses* that the terminal's ACS support is broken; the library then outputs Unicode code points that correspond to the forms-drawing characters. Set it to zero (or a non-integer) to disable the special check for terminal type names matching "linux" or "screen", directing *ncurses* to assume that the ACS feature works if the terminal type description advertises it.

As an alternative to use of this variable, *ncurses* checks for an extended *terminfo* numeric capability **U8** that can be compiled using "**tic -x**". Examples follow.

```
# linux console, if patched to provide working
# VT100 shift-in/shift-out, with corresponding font.
linux-vt100|linux console with VT100 line-graphics,
    U8#0, use=linux,

# xterm with vt100Graphics resource set to false
xterm-utf8|xterm relying on UTF-8 line-graphics,
    U8#1, use=xterm,
```

The two-character name "U8" was chosen to permit its use via *ncurses*'s *termcap* interface.

NCURSES_TRACE

At initialization, *ncurses* (in its debugging configuration) checks for this variable's presence. If defined with an integral value, the library calls **curSES_trace(3X)** with that value as the argument.

TERM

The *TERM* variable denotes the terminal type. Each is distinct, though many are similar. It is commonly set by terminal emulators to help applications find a workable terminal description. Some choose a popular approximation such as “ansi”, “vt100”, or “xterm” rather than an exact fit to their capabilities. Not infrequently, an application will have problems with that approach; for example, a key stroke may not operate correctly, or produce no effect but seeming garbage characters on the screen.

Setting *TERM* has no effect on hardware operation; it affects the way applications communicate with the terminal. Likewise, as a general rule (*xterm(1)* being a rare exception), terminal emulators that allow you to specify *TERM* as a parameter or configuration value do not change their behavior to match that setting.

TERMCAP

If *ncurses* is configured with *termcap* support, it checks for a terminal type description in *termcap* format if one in *terminfo* format is not available. Setting this variable directs *ncurses* to ignore the usual *termcap* database location, */etc/termcap*; see *TERMPATH* below. *TERMCAP* should contain either a terminal description (with newlines stripped out), or a file name indicating where the information required by the *TERM* environment variable is stored.

TERMINFO

ncurses can be configured to read terminal type description databases in various locations using different formats. This variable overrides the default location.

- Descriptions in *terminfo* format are normally stored in a directory tree using subdirectories named for the common first letters of the terminal types named therein. System V used this scheme.
- If *ncurses* is configured to use hashed databases, then *TERMINFO* may name its location, such as */usr/share/terminfo.db*, rather than */usr/share/terminfo/*.

The hashed database uses less disk space and is a little faster than the directory tree. However, some applications assume the existence of the directory tree and read it directly, ignoring the *terminfo* API.

- If *ncurses* is configured with *termcap* support, this variable may contain the location of a *termcap* file.
- If the value of *TERMINFO* begins with “hex:” or “b64:”, *ncurses* uses the remainder of the value as a compiled *terminfo* description. You might produce the base64 format using **infocmp(1M)**.

```
TERMINFO=$(infocmp -0 -Q2 -q)
export TERMINFO
```

ncurses uses the compiled description only if it corresponds to the terminal type identified by *TERM*.

Setting *TERMINFO* is the simplest, but not the only, way to direct *ncurses* to a terminal database. The search path is as follows.

- the last terminal database to which the running *ncurses* application wrote, if any
- the location specified by the *TERMINFO* environment variable
- *\$HOME/terminfo*
- locations listed in the *TERMINFO_DIRS* environment variable
- location(s) configured and compiled into *ncurses*
 - */etc/terminfo:/usr/share/terminfo*
 - */usr/share/terminfo*

TERMINFO_DIRS

This variable specifies a list of locations, akin to *PATH*, in which *ncurses* searches for the terminal type descriptions described by *TERMINFO* above. The list items are separated by colons on Unix and semicolons on OS/2 EMX. System V *terminfo* lacks a corresponding feature; *TERMINFO_DIRS* is an *ncurses*

extension.

TERMPATH

If *TERMCAP* does not hold a terminal type description or file name, then *ncurses* checks the contents of *TERMPATH*, a list of locations, akin to *PATH*, in which it searches for *termcap* terminal type descriptions. The list items are separated by colons on Unix and semicolons on OS/2 EMX.

If both *TERMCAP* and *TERMPATH* are unset or invalid, *ncurses* searches for the files */etc/termcap*, */usr/share/misc/termcap*, and *\$HOME/.termcap*, in that order.

ALTERNATE CONFIGURATIONS

Many different *ncurses* configurations are possible, determined by the options given to the *configure* script when building the library. Run the script with the **--help** option to peruse them all. A few are of particular significance to the application developer employing *ncurses*.

--disable-overwrite

Avoid file name conflicts between *ncurses* and an existing *curses* installation on the system. The standard C preprocessor inclusion for the *curses* library is as follows.

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

If *ncurses* is installed disabling overwrite, it puts its header files in a subdirectory. Here is an example.

```
#include <ncurses/curses.h>
```

With **--disable-overwrite**, installation also omits a symbolic link that would cause the compiler's **-lcurses** option to link object files with *ncurses* instead of the system *curses* library.

The directory used by this configuration of *ncurses* is shown in section "SYNOPSIS" above.

--enable-widec

(default for ABI 6+ since 2023-10-21) Enable support for wide characters. The *configure* script renames the *ncurses* library (and the *tinio* library, if **--with-termliib** is also specified), appending "w". An application desirous of wide-character support then uses **-lncursesw** (or **-ltinw**) instead of **-lcurses** (or **-ltinfo**) as its linker option. The *ncurses++*, *panel*, *form*, and *menu* libraries are renamed similarly.

An application must also define certain C preprocessor symbols to enable wide-character features in *curses* header files to use the extended (wide-character) functions. The symbol that enables these features has changed since X/Open Curses Issue 4.

- Originally, the wide-character feature required the symbol `_XOPEN_SOURCE_EXTENDED`, but that was valid only for XPG4 (1996).
- Later, that was deemed conflicting with an `_XOPEN_SOURCE` value of 500.
- As of mid-2018, no *ncurses* feature requires a `_XOPEN_SOURCE` value greater than 600. However, X/Open Curses Issue 7 (2009) recommends defining it to 700.
- Alternatively, you can enable the feature by defining **NCURSES_WIDECHAR** with the caveat that some header file other than *curses.h* may require a specific value for `_XOPEN_SOURCE` (or a system-specific symbol).

The *curses.h* header file installed for the wide-character library is designed to be compatible with the non-wide library's header. Only the size of the `WINDOW` structure differs; few applications require more than pointers to `WINDOW`.

If *ncurses*'s header files are installed allowing overwrite (the default, but see **--disable-overwrite** above), the wide-character library's headers should be installed last by packaging systems and similar, to allow applications to be built using either library from the same set of headers.

--with-pthread

Enable support for multi-threaded applications. The *configure* script renames the *ncurses* library (and the *tinio* library, if **--with-termliib** is also specified), appending "t" (before any "w" added by

--enable-widex). An application desirous of support for multiple threads of execution then uses, for example, **-lncursesw** (or **-ltinfo**) instead of **-lncurses** (or **-ltinfo**) as its linker option. The *ncurses++*, *panel*, *form*, and *menu* libraries are renamed similarly.

ncursesw and *ncursesw* replace global variables such as **LINES** with macros allowing read-only access. At the same time, they provide functions to set these values. Very few applications require changes to work with this convention.

--with-shared

--with-normal

--with-debug

--with-profile

Mandate compilation of the *ncurses* library (and the *tinio* library, if **--with-termli** is also specified) in the specified forms. The shared and normal (static) library names differ by their suffixes, as with *libncurses.so* and *libncurses.a*. The debugging and profiling libraries add a “_g” and a “_p” to the roots of these respective names, forming *libncurses_g.so* and *libncurses_p.a*, for example. The *ncurses++*, *panel*, *form*, and *menu* libraries are made available similarly.

--with-termli

Provide *ncurses*’s lower-level terminal interface functions (those that do not depend on the *SCREEN* and *WINDOW* abstractions) in a library named *tinio*. This arrangement reduces an application’s linking and/or loading times when it does not require *curses*’s higher-level features.

The following pages document *curses* functions provided by *tinio*.

- **curs_extend(3X)** – miscellaneous *ncurses* extensions
- **curs_inopts(3X)** – *curses* input options
- **curs_kernel(3X)** – low-level *curses* routines
- **curs_termattr(3X)** – *curses* environment query routines
- **curs_termcap(3X)** – *curses* emulation of *termcap*
- **curs_terminfo(3X)** – *curses* interface to *terminfo* database
- **curs_util(3X)** – miscellaneous *curses* utility routines

--with-trace

Expose the **curses_trace(3X)** function in the *ncurses(w)* shared and static libraries. Normally, it is available only in the debugging library. (If **--with-termli** is also specified, *tinio(w)* rather than *ncurses(w)* supplies the deprecated **trace(3X)** function.) An application’s configuration script should check for the function’s existence rather than assuming its confinement to the debugging library.

FILES

/usr/share/tabset
tab stop initialization database

/usr/share/terminfo
compiled terminal capability database

NOTES

X/Open Curses permits most functions it specifies to be made available as macros as well. *ncurses* does so

- for functions that return values via their parameters,
- to support obsolete features,
- to reuse functions (for example, those that move the cursor before another operation), and
- in a few special cases.

If the standard output file descriptor of an *ncurses* program is redirected to something that is not a terminal device, the library writes screen updates to the standard error file descriptor. This was an undocumented feature of SVr3 *curses*.

See subsection “Header Files” below regarding symbols exposed by inclusion of *curses.h*.

EXTENSIONS

ncurses enables an application to capture mouse events from certain terminals, including *xterm*(1); see **curs_mouse**(3X).

ncurses provides a means of responding to window resizing events, as when running in a GUI terminal emulator application such as *xterm*; see **resizeterm**(3X) and **wresize**(3X).

ncurses allows an application to query the terminal for the presence of a wide variety of special keys; see **has_key**(3X).

ncurses extends the fixed set of function key capabilities specified by X/Open Curses by allowing the application programmer to define additional key events at runtime; see **define_key**(3X), **key_defined**(3X), **keybound**(3X), and **keyok**(3X).

ncurses can exploit the capabilities of terminals implementing ISO 6429/ECMA-48 SGR 39 and SGR 49 sequences, which allow an application to reset the terminal to its original foreground and background colors. From a user’s perspective, the application is able to draw colored text on a background whose color is set independently, providing better control over color contrasts. See **default_colors**(3X).

An *ncurses* application can eschew knowledge of *SCREEN* and *WINDOW* structure internals, instead using accessor functions such as **is_cbreak**(3X) and **is_scrollback**(3X).

ncurses enables an application to direct its output to a printer attached to the terminal device; see **curs_print**(3X).

ncurses offers **slk_attr**(3X) as a counterpart of **attr_get**(3X) for soft-label key lines, and **extended_slk_color**(3X) as a form of **slk_color**(3X) that can gather color information from them when many colors are supported.

ncurses permits modification of **unctrl**(3X)’s behavior; see **use_legacy_coding**(3X).

Rudimentary support for multi-threaded applications may be available; see **curs_threads**(3X).

Functions that ease the management of multiple screens can be exposed; see **curs_sp_funcs**(3X).

To aid applications to debug their memory usage, *ncurses* optionally offers functions to more aggressively free memory it dynamically allocates itself; see **curs_memleaks**(3X).

The library facilitates auditing and troubleshooting of its behavior; see **curs_trace**(3X).

Compiling *ncurses* with the option **-DUSE_GETCAP** causes it to fall back to reading */etc/termcap* if the terminal setup code cannot find a *terminfo* entry corresponding to *TERM*. Use of this feature is not recommended, as it essentially includes an entire *termcap* compiler in the *ncurses* startup code, at a cost in memory usage and application launch latency.

PDCCurses and NetBSD *curses* incorporate some *ncurses* extensions. Individual man pages indicate where this is the case.

PORTABILITY

X/Open Curses defines two levels of conformance, “base” and “enhanced”. The latter includes several additional features, such as wide-character and color support. *ncurses* intends base-level conformance with X/Open Curses, and supports all features of its enhanced level except the *untic* utility.

Differences between X/Open Curses and *ncurses* are documented in the “PORTABILITY” sections of applicable man pages.

Error Checking

In many cases, X/Open Curses is vague about error conditions, omitting some of the SVr4 documentation.

Unlike other implementations, *ncurses* checks pointer parameters, such as those to *WINDOW* structures, to ensure that they are not null. This is done primarily to guard against programmer error. The standard interface does not provide a way for the library to tell an application which of several possible errors occurred. An application that relies on *ncurses* to check its function parameters for validity limits its portability and robustness.

Padding Differences

In historical *curses* implementations, delays embedded in the *terminfo* capabilities **carriage_return** (**cr**), **scroll_forward** (**ind**), **cursor_left** (**cub1**), **form_feed** (**ff**), and **tab** (**ht**) activated corresponding delay bits in the Unix terminal driver. *ncurses* performs all padding by sending NUL bytes to the device. This method is slightly more expensive, but narrows the interface to the Unix kernel significantly and correspondingly increases the package's portability.

Header Files

The header file *curses.h* itself includes the header files *stdio.h* and *unctrl.h*.

X/Open Curses has more to say,

The inclusion of *curses.h* may make visible all symbols from the headers *stdio.h*, *term.h*, *termios.h*, and *wchar.h*.

but does not finish the story. A more complete account follows.

- The first *curses*, in 4BSD, provided a *curses.h* file.

BSD *curses* code included *curses.h* and *unctrl.h* from an internal header file *curses.ext*, where “ext” abbreviated “externs”.

The implementations of *printw* and *scanw* used undocumented internal functions of the standard I/O library (*_doprnt* and *_doscan*), but nothing in *curses.h* itself relied upon *stdio.h*.

- SVr2 *curses* added *newterm*, which relies upon *stdio.h* because its function prototype employs the *FILE* type.

SVr4 *curses* added *putwin* and *getwin*, which also use *stdio.h*.

X/Open Curses specifies all three of these functions.

SVr4 *curses* and X/Open Curses do not require the developer to include *stdio.h* before *curses.h*. Both document use of *curses* as requiring only *curses.h*.

As a result, standard *curses.h* always includes *stdio.h*.

- X/Open Curses and SVr4 *curses* are inconsistent with respect to *unctrl.h*.

As noted in **curl_util**(3X), *ncurses* includes *unctrl.h* from *curses.h* (as SVr4 does).

- X/Open Curses's comments about *term.h* and *termios.h* may refer to HP-UX and AIX.

HP-UX *curses* includes *term.h* from *curses.h* to declare *setupterm* in *curses.h*, but *ncurses* and Solaris *curses* do not.

AIX *curses* includes *term.h* and *termios.h*. Again, *ncurses* and Solaris *curses* do not.

- X/Open Curses says that *curses.h* **may** include *term.h*, but does not require it to do so.

Some programs use functions declared in both *curses.h* and *term.h*, and must include both header files in the same translation unit.

The header files supplied by *ncurses* include the standard library headers required for its declarations, so *ncurses*'s own header files can be included in any order. For portability even to old AIX systems, include *curses.h* before *term.h*.

- X/Open Curses says “may make visible” because including a header file does not necessarily make visible all of the symbols in it (consider **#ifdef** and similar).

For instance, *ncurses*'s *curses.h* **may** include *wchar.h* if the proper symbol is defined, and if *ncurses* is configured for wide-character support. If *wchar.h* is included, its symbols **may** be made visible depending on the value of the *_XOPEN_SOURCE* feature test macro.

- X/Open Curses mandates an application's inclusion of one standard C library header in a special case: *stdarg.h* before *curses.h* to prototype the functions *vw_printw* and *vw_scanw* (as well as the obsolete *vwprintw* and *vwscanw*). Each of these takes a variadic argument list, a *va_list* parameter, like that of *printf*(3).

SVr3 *curses* introduced the two obsolete functions, and X/Open Curses the others. In between, SVr4 *curses* provided for the possibility that an application might include either *varargs.h* or *stdarg.h*. These represented contrasting approaches to handling variadic argument lists. The older interface, *varargs.h*, used a pointer to *char* for variadic functions' *va_list* parameter. Later, the list acquired its own standard data type, *va_list*, defined in *stdarg.h*, empowering the compiler to check the types of a function call's actual parameters against the formal ones declared in its prototype.

No conforming implementations of X/Open Curses require an application to include *stdarg.h* before *curses.h* because they either have allowed for a special type, or, like *ncurses*, they include *stdarg.h* themselves to provide a portable interface.

AUTHORS

Zeyd M. Ben-Halim, Eric S. Raymond, Thomas E. Dickey. Based on *ncurses* by Pavel Curtis.

SEE ALSO

curs_variables(3X), **terminfo(5)**, **user_caps(5)**