NAME

getch, **wgetch**, **mvgetch**, **mvwgetch**, **ungetch**, **has_key** − get (or push back) characters from *curses* terminal keyboard buffer

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);

int ungetch(int c);

/* extension */
int has_key(int c);
```

DESCRIPTION

Reading Characters

wgetch gathers a key event from the terminal keyboard associated with a *curses* window *win*. **ncurses**(3X) describes the variants of this function.

When input is pending, **wgetch** returns an integer identifying the key event; for alphanumeric and punctuation keys, the space bar, and (usually) the Backspace, Tab, Return, and Escape keys, this value corresponds to the character encoding used by the terminal. Use of the control key as a modifier, by holding it down while pressing and releasing another key, often results in a distinct code. The behavior of other keys depends on whether *win* is in keypad mode; see subsection "Keypad Mode" below.

If no input is pending, then if the no-delay flag is set in the window (see **nodelay**(3X)), the function returns **ERR**; otherwise, *curses* waits until the terminal has input. If **cbreak**(3X) or **raw**(3X) has been called, this happens after *curses* reads one key event. If**nocbr eak**(3X) or **noraw**(3X) has been called, it occurs when *curses* reads a newline. (Because the terminal's canonical or "cooked" mode is line-buffered, multiple **wgetch** calls may then be necessary to empty the input queue.) If **halfdelay**(3X) has been called, *curses* waits until input is available or the specified delay elapses.

If **echo**(3X) has been called, and the window is not a pad, *curses* writes the returned character *c* to the window (at the cursor position) per the following rules.

- If *c* matches the terminal's erase character (see **erasechar**(3X)), and the cursor is not at the window's leftmost column, the cursor moves leftward one position and the new position is erased as if **wmove**(3X) and then **wdelch**(3X) were called. When the window's keypad mode is enabled (see below), **KEY_LEFT** and **KEY_BACKSPACE** are handled the same way.
- *curses* writes any other *c* to the window, as with **wechochar**(3X).
- If the window *win* has been moved or modified since the last call to **wrefresh**(3X), *curses* calls **wrefresh** on it.

If *c* is a carriage return and **nl**(3X) has been called, **wgetch** returns the character code for line feed instead.

Keypad Mode

Call **keypad**(3X) on a window to configure keypad mode when reading input from it. In *keypad mode*, *curses* treats key strokes not from the alphabetic section of the keyboard (those corresponding to the ECMA-6 character set — see *ascii*(7) — optionally modified by either the control or shift keys) as *function* keys. (In*cur ses*, the term "function key" includes but is not limited to keycaps engraved with "F1", "PF1", and so on.) If a window is in keypad mode, **wgetch** translates these key strokes to a numeric code corresponding to the **KEY_** symbols listed in subsection "Key Codes" below. If the window is not in keypad mode, the input queue populates with the characters of the function key's escape sequence, which the application must collect individually with multiple **wgetch** calls.

- The *curses.h* header file declares many *function keys* whose names begin with **KEY_**; these object-like macros have integer values outside the range of eight-bit character codes.

- In *ncurses*, *user-defined function keys* are configured with **define_key**(3X); they have no names, but are also expected to have integer values outside the range of eight-bit character codes.

A variable intended to hold a function key code must thus be of type *short* or larger.

Most terminals one encounters follow the ECMA-48 standard insofar as their function keys produce character sequences prefixed with the escape character ESC. This fact implies that *curses* cannot distinguish a user's press of the escape key (assuming it sends ESC) from the beginning of a function key's character sequence without waiting to see if, and how soon, further input arrives.

- If the escape sequence matches a string capability defining a function key for the terminal type (such as **key_home** (**khome**) or **key_up** (**kuu1**)), **wgetch** returns the function key code corresponding to the unique sequence defined by the terminal.
- If the escape sequence matches no function keys defined for the terminal type, call **wgetch** repeatedly to obtain the codes of the individual characters of the sequence, in the order they occurred in the input.
- If **wgetch** cannot decide the validity of the input as a function key because it has not read enough characters to disambiguate it, the function waits until it has this information or the *escape delay* elapses. Configure the escape delay with the global variable **ESCDELAY**, an extension (see section "EXTENSIONS" below), or the environment variable of the same name (see section "ENVIRONMENT" of**ncurses**(3X)), also an extension.

Consequently, a user of a *curses* application that employs keypad mode may experience a pause or "hang" after pressing the escape key while *curses* collects sufficient characters to disambiguate the input. If the window is in "no time-out" mode, the escape delay is effectively infinite; see **notimeout**(3X). In the event of such a pause, further typing "awakens" *curses*.

Ungetting Characters

ungetch places *c* into the input queue to be returned by the next call to **wgetch**. A single input queue serves all windows associated with the screen.

Key Codes

The header file *curses.h* defines the following function key codes.

- Except for the special case of **KEY_RESIZE**, a window's keypad mode must be enabled for **wgetch** to read these codes from it.
- Not all of these are necessarily supported on any particular terminal.
- The naming convention may seem obscure, with some apparent misspellings (such as "RSUME" for "resume"); the names correspond to the *terminfo* capability names for the keys, and were standardized before the IBM PC/AT keyboard layout achieved a dominant position in industry.

Symbol	Key name
KEY_BREAK	Break key
KEY_DOWN **KEY_UP** **KEY_LEFT** **KEY_RIGHT**	Arrow keys
KEY_HOME	Home key (upward+left arrow)
KEY_BACKSPACE	Backspace
KEY_F0	Function keys; space for 64 keys is reserved
KEY_F(*n*)	Function key *n* where $0 \le n \le 63$
KEY_DL	Delete line
KEY_IL	Insert line
KEY_DC	Delete character
KEY_IC	Insert character/Enter insert mode
KEY_EIC	Exit insert character mode
KEY_CLEAR	Clear screen

KEY_EOS	Clear to end of screen
KEY_EOL	Clear to end of line
KEY_SF	Scroll one line forward
KEY_SR	Scroll one line backward (reverse)
KEY_NPAGE	Next page/Page up
KEY_PPAGE	Previous page/Page down
KEY_STAB	Set tab
KEY_CTAB	Clear tab
KEY_CATAB	Clear all tabs
KEY_ENTER	Enter/Send
KEY_SRESET	Soft (partial) reset
KEY_RESET	(Hard) reset
KEY_PRINT	Print/Copy
KEY_LL	Home down/Bottom (lower left)
KEY_A1	Upper left of keypad
KEY_A3	Upper right of keypad
KEY_B2	Center of keypad
KEY_C1	Lower left of keypad
KEY_C3	Lower right of keypad
KEY_BTAB	Back tab key
KEY_BEG	Beg(inning) key
KEY_CANCEL	Cancel key
KEY_CLOSE	Close key
KEY_COMMAND	Cmd (command) key
KEY_COPY	Copy key
KEY_CREATE	Create key
KEY_END	End key
KEY_EXIT	Exit key
KEY_FIND	Find key
KEY_HELP	Help key
KEY_MARK	Mark key
KEY_MESSAGE	Message key
KEY_MOUSE	Mouse event occurred
KEY_MOVE	Move key
KEY_NEXT	Next object key
KEY_OPEN	Open key
KEY_OPTIONS	Options key
KEY_PREVIOUS	Previous object key
KEY_REDO	Redo key
KEY_REFERENCE	Ref(erence) key
KEY_REFRESH	Refresh key
KEY_REPLACE	Replace key
KEY_RESIZE	Screen resized
KEY_RESTART	Restart key
KEY_RESUME	Resume key
KEY_SAVE	Save key
KEY_SELECT	Select key
KEY_SUSPEND	Suspend key
KEY_UNDO	Undo key
KEY_SBEG	Shifted beginning key
KEY_SCANCEL	Shifted cancel key
KEY_SCOMMAND	Shifted command key
KEY_SCOPY	Shifted copy key

KEY_SCREATE	Shifted create key
KEY_SDC	Shifted delete character key
KEY_SDL	Shifted delete line key
KEY_SEND	Shifted end key
KEY_SEOL	Shifted clear line key
KEY_SEXIT	Shifted exit key
KEY_SFIND	Shifted find key
KEY_SHELP	Shifted help key
KEY_SHOME	Shifted home key
KEY_SIC	Shifted insert key
KEY_SLEFT	Shifted left arrow key
KEY_SMESSAGE	Shifted message key
KEY_SMOVE	Shifted move key
KEY_SNEXT	Shifted next object key
KEY_SOPTIONS	Shifted options key
KEY_SPREVIOUS	Shifted previous object key
KEY_SPRINT	Shifted print key
KEY_SREDO	Shifted redo key
KEY_SREPLACE	Shifted replace key
KEY_SRIGHT	Shifted right arrow key
KEY_SRSUME	Shifted resume key
KEY_SSAVE	Shifted save key
KEY_SSUSPEND	Shifted suspend key
KEY_SUNDO	Shifted undo key

Many keyboards feature a nine-key directional pad.

A1	up	A3
left	B2	right
C1	down	C3

Two of the symbols in the list above do *not* correspond to a physical key.

- **wgetch** returns **KEY_RESIZE**, even if the window's keypad mode is disabled, if *ncurses* has handled a *SIGWINCH* signal since **wgetch** was called; see **initscr**(3X) and **resizeterm**(3X).
- **wgetch** returns **KEY_MOUSE** to indicate that a mouse event is pending collection; see **curs_mouse**(3X). Receipt of this code requires a window's keypad mode to be enabled, because to interpret mouse input (as with *xterm*(1)'s mouse protocol), *ncurses* must read an escape sequence, as with a function key.

Testing Key Codes

In *ncurses*, **has_key** returns a Boolean value indicating whether the terminal type recognizes its parameter as a key code value. See also**define_k ey**(3X) and **key_defined**(3X).

RETURN VALUE

wgetch returns a key code identifying the key event as described above, which may include **KEY_RESIZE** or **KEY_MOUSE** indicating non-key events, or **ERR** on failure. **wgetch** fails if its timeout expires without any data arriving, which cannot happen if **nodelay**(3X) is in effect on the window.

In *ncurses*, **wgetch** also fails if

- the *curses* screen has not been initialized,
- (for functions taking a *WINDOW* pointer argument) *win* is a null pointer, or
- execution was interrupted by a signal, in which case the library sets *errno* to *EINTR*.

Functions prefixed with "mv" first perform cursor movement and fail if the position (*y*, *x*) is outside the window boundaries.

ungetch returns **OK** on success and **ERR** on failure. In*ncur ses*, **ungetch** fails if

- the *curses* screen has not been initialized, or
- there is no more room in the input queue.

has_key returns **TRUE** or **FALSE**.

NOTES

getch, **mvgetch**, and **mvwgetch** may be implemented as macros.

curses discourages assignment of the ESC key to a discrete function by the programmer because the library requires a delay while it awaits the potential remainder of a terminal escape sequence.

Some key strokes are indistinguishable from control characters; for example, **KEY_ENTER** may be the same as **^M**, and **KEY_BACKSPACE** may be the same as **^H** or **^?**. Consult the*terminfo* entry for the terminal type to determine whether this is the case; see **infocmp**(1). Some*cur ses* implementations, including *ncurses*, honor the *terminfo* key definitions; others treat such control characters specially.

curses distinguishes the Enter keys in the alphabetic and numeric keypad sections of a keyboard because (most) terminals do. **KEY_ENTER** refers to the k ey on the numeric keypad and, like other function keys, is reliably recognized only if the window's keypad mode is enabled.

- The *terminfo* **key_enter** (**kent**) capability describes the character (sequence) sent by the Enter key of a terminal's numeric (or similar) keypad.
- "Enter or send" is X/Open Curses's description of this key.

curses treats the Enter or Return key in the *alphabetic* section of the keyboard differently.

- It usually produces a control code for carriage return (**^M**) or line feed (**^J**).
- Depending on the terminal mode (raw, cbreak, or canonical), and whether **nl**(3X) or **nonl**(3X) has been called, **wgetch** may return either a carriage return or line feed upon an Enter or Return key stroke.

Use of **wgetch** with **echo**(3X) and neither **cbreak**(3X) nor **raw**(3X) is not well-defined.

Historically, the list of key code macros above was influenced by the keyboard of the AT&T 7300 (also known variously as the "3B1", "Safari 4", and "UNIX PC"), a 1985 machine rich in function keys. Today's computer keyboards are based on that of the IBM PC/AT and tend to have fewer. A*cur ses* application can expect such a keyboard to transmit key codes **KEY_UP**, **KEY_DOWN**, **KEY_LEFT**, **KEY_RIGHT**, **KEY_HOME**, **KEY_END**, **KEY_PPAGE** (Page Up), **KEY_NPAGE** (Page Down), **KEY_IC** (Insert), **KEY_DC** (Delete), **KEY_A1**, **KEY_A3**, **KEY_B2**, **KEY_C1**, **KEY_C3**, and **KEY_F(***n***)** for $1 \le n \le 12$.

EXTENSIONS

In *ncurses*, when a window's "no time-out" mode is *not* set, the **ESCDELAY** variable configures the duration of the timer used to disambiguate a function key character sequence from a series of key strokes beginning with ESC typed by the user; see **curs_variables**(3X).

has_key is an *ncurses* extension, and is not found in SVr4 *curses*, 4.4BSD *curses*, or any other previous *curses* implementation.

PORTABILITY

Applications employing *ncurses* extensions should condition their use on the visibility of the **NCURSES_VERSION** preprocessor macro.

Except as noted in section "EXTENSIONS" above, X/Open Curses Issue 4 describes these functions. It specifies no error conditions for them.

SVr4 describes a successful return value only as "an integer value other than *ERR*".

wgetch reads only single-byte characters.

The echo behavior of these functions on input of *KEY_* or backspace characters is not documented in SVr4 *curses*.

The behavior of *wgetch* in the presence of signal handlers is not documented in SVr4 *curses* and is unspecified by X/Open Curses. In historical *curses* implementations, it varied depending on whether the operating

system's dispatch of a signal to a handler interrupted a *read*(2) call in progress, and also (in some implementations) whether an input timeout or non-blocking mode had been set. A portable *curses* application prepares for two cases: (a) signal receipt does not interrupt *wgetch*; and (b) signal receipt interrupts *wgetch* and causes it to return *ERR* with *errno* set to *EINTR*.

KEY_MOUSE is mentioned in X/Open Curses, along with a few related *terminfo* capabilities, but no higher-level functions use the feature. The implementation in *ncurses* is an extension.

KEY_RESIZE and *has_key* are extensions first implemented for *ncurses*. By 2022,*PDCur ses* and NetBSD *curses* had added them along with *KEY_MOUSE*.

HISTORY

4BSD (1980) introduced *wgetch* and its variants.

SVr3 (1987) added *ungetch*.

ncurses 1.9.9g (1996) furnished the *has_key* extension.

SEE ALSO

ECMA-6 "7-bit coded Character Set" <https://ecma−international.org/publications−and−standards/standards/ecma−6/>

ECMA-48 "Control Functions for Coded Character Sets" <https://ecma−international.org/publications−and−standards/standards/ecma−48/>

curs_get_wch(3X) describes comparable functions of the *ncurses* library in its wide-character configuration (*ncursesw*).

curses(3X), **curs_addch**(3X), **curs_inopts**(3X), **curs_mouse**(3X), **curs_move**(3X), **curs_outopts**(3X), **curs_refresh**(3X), **curs_variables**(3X), **resizeterm**(3X), **ascii**(7)