NAME

cbreak, **echo**, **halfdelay**, **intrflush**, **is_cbreak**, **is_echo**, **is_nl**, **is_raw**, **keypad**, **meta**, **nl**, **nocbreak**, **nodelay**, **noecho**, **nonl**, **noqiflush**, **noraw**, **notimeout**, **qiflush**, **raw**, **timeout**, **wtimeout**, **typeahead** - get and set *curses* terminal input options

SYNOPSIS

```
#include <curses.h>

int cbreak(void);
int nocbreak(void);

int echo(void);
int noecho(void);

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 nl(void);
int nonl(void);

int raw(void);
int noraw(void);

void qiflush(void);
void noqiflush(void);

int halfdelay(int tenths);
void timeout(int delay);
void wtimeout(WINDOW *win, int delay);

int typeahead(int fd);

/* extensions */
int is_cbreak(void);
int is_echo(void);
int is_nl(void);
int is_raw(void);
```

DESCRIPTION

ncurses provides several functions that let an application change the way input from the terminal is handled. Some are global, applying to all windows. Others apply only to a specific window. Window-specific settings are not automatically applied to new or derived windows. An application must apply these to each window if the same behavior is desired.

cbreak, nocbreak

Normally, the terminal driver buffers typed characters until a newline or carriage return is typed. The **cbreak** routine disables line buffering and erase/kill character-processing (interrupt and flow control characters are unaffected), making characters typed by the user immediately available to the program. The **nocbreak** routine returns the terminal to normal (cooked) 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**(3X) for a discussion of how these routines interact with **echo** and **noecho**.]

echo, noecho

The **echo** and **noecho** routines control whether characters typed by the user are echoed by **getch**(3X) as they are typed. Echoing by the terminal 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**(3X) for a discussion of how these routines interact with **cbreak** and **nocbreak**.]

halfdelay

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 **nocbreak** to leave half-delay mode.

intrflush

If the **intrflush** option is enabled (*bf* is **TRUE**), and an interrupt key is pressed on the keyboard (interrupt, break, quit), all output in the terminal driver queue is 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 the option (*bf* is **FALSE**), prevents the flush. The default for the option is inherited from the terminal driver settings. The *win* argument is ignored.

keypad

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**(3X) 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**(3X) is called. The default value for keypad is **FALSE**.

meta

Initially, whether the terminal returns 7 or 8 significant bits on input depends on the control mode of the terminal driver [see *termios*(3)]. To force 8 bits to be returned, invoke **meta**(*win*, **TRUE**); this is equivalent, under POSIX, to setting the CS8 flag on the terminal. To force 7 bits to be returned, invoke **meta**(*win*, **FALSE**); this is equivalent, under POSIX, to setting the CS7 flag on the terminal. 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.

nl, nonl

The **nl** and **nonl** routines control whether the underlying display device translates the return key into newline on input.

nodelay

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.

notimeout

When interpreting an escape sequence, **wgetch**(3X) 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 distinguish sequences produced by a function key from those typed by a user.

raw, noraw

The **raw** and **noraw** routines place the terminal 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 terminal driver that are not set by *curses*.

qiflush, nqiflush

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 **termios**(3)]. When **qiflush** is called, the queues will be flushed when these control characters are read. You may want to call **noqiflush** in a signal handler if you want output to continue as though the interrupt had not occurred, after the handler exits.

timeout, wtimeout

The **timeout** and **wtimeout** routines set blocking or non-blocking read for a given window. If *delay* is negative, a blocking read is used (i.e., waits indefinitely for input). If *delay* is zero, then a non-blocking read is used (i.e., *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).

typeahead

curses does "line-breakout optimization" by looking for typeahead periodically while updating the screen. If input is found, and it is coming from a terminal, the current update is postponed until **refresh**(3X) 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 *fd* is to be used to check for typeahead instead. If *fd* is -1, then no typeahead checking is done.

RETURN VALUE

All routines that return an integer return **ERR** upon failure and **OK** (SVr4 specifies only "an integer value other than **ERR**") upon successful completion, unless otherwise noted in the preceding routine descriptions.

X/Open Curses does not specify any error conditions. In this implementation, functions with a window parameter will return an error if it is null. Any function will also return an error if the terminal was not initialized. Also,

halfdelay

returns an error if its parameter is outside the range 1..255.

NOTES

echo, **noecho**, **halfdelay**, **intrflush**, **meta**, **nl**, **nonl**, **nodelay**, **notimeout**, **noqiflush**, **qiflush**, **timeout**, and **wtimeout** may be implemented as macros.

noraw and **nocbreak** follow historical practice in that they attempt to restore normal ("cooked") mode from raw and cbreak modes respectively. Mixing **raw**/**noraw** and **cbreak**/**nocbreak** calls leads to terminal driver control states that are hard to predict or understand; doing so is not recommended.

EXTENSIONS

ncurses provides four "is_" functions that may be used to detect if the corresponding flags were set or reset.

Query	Set	Reset
is_cbreak	cbreak	nocbreak
is_echo	echo	noecho
is_nl	nl	nonl
is_raw	raw	noraw

In each case, the function returns

1 if the flag is set,

0 if the flag is reset, or

-1 if the library is not initialized.

They were designed for **ncurses**(3X), and are 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, Version 2 describes these functions.

ncurses follows X/Open Curses and the historical practice of AT&T *curses* implementations, in that the echo bit is cleared when *curses* initializes the terminal state. BSD *curses* differed from this slightly; it left the echo bit on at initialization, but the BSD **raw** call turned it off as a side effect. For best portability, set **echo** or **noecho** explicitly just after initialization, even if your program remains in cooked mode.

X/Open Curses is ambiguous regarding whether **raw** should disable the CR/LF translations controlled by **nl** and **nonl**. BSD *curses* did turn off these translations; AT&T *curses* (at least as late as SVr1) did not. *ncurses* does so, on the assumption that a programmer requesting raw input wants a clean (ideally, 8-bit clean) connection that the operating system will not alter.

When **keypad** is first enabled, *ncurses* loads the key definitions for the current terminal description. If the terminal description includes extended string capabilities, e.g., from using the **-x** option of **tic**, then *ncurses* also defines keys for the capabilities whose names begin with "k". The corresponding keycodes are generated and (depending on previous loads of terminal descriptions) may differ from one

execution of a program to the next. The generated keycodes are recognized by the **keyname**(3X) function (which will then return a name beginning with "k" denoting the terminfo capability name rather than "K", used for *curses* key names). On the other hand, an application can use **define_key**(3X) to establish a specific keycode for a given string. This makes it possible for an application to check for an extended capability's presence with **tigetstr**, and reassign the keycode to match its own needs.

Low-level applications can use **tigetstr** to obtain the definition of any particular string capability. Higher-level applications which use the *curses* **wgetch** and similar functions to return keycodes rely upon the order in which the strings are loaded. If more than one key definition has the same string value, then **wgetch** can return only one keycode. Most *curses* implementations (including *ncurses*) load key definitions in the order defined by the array of string capability names. The last key to be loaded determines the keycode which will be returned. In *ncurses*, you may also have extended capabilities interpreted as key definitions. These are loaded after the predefined keys, and if a capability's value is the same as a previously-loaded key definition, the later definition is the one used.

HISTORY

Formerly, *ncurses* used **nl** and **nonl** to control the conversion of newlines to carriage return/line feed on output as well as input. X/Open Curses documents the use of these functions only for input. This difference arose from converting the *pcurses* source (1986), which used *ioctl*(2) calls and the *sgttyb* structure, to *termios* (the POSIX terminal API). In the former, both input and output were controlled via a single option **CRMOD**, while the latter separates these features. Because that conversion interferes with output optimization, *ncurses* 6.2 (2020) amended **nl** and **nonl** to eliminate their effect on output.

SEE ALSO

curses(3X), **curs_getch**(3X), **curs_initscr**(3X), **curs_util**(3X), **define_key**(3X), **termios**(3)