

mouse(3NCURSES)

mouse(3NCURSES)

**NAME**

**has\_mouse**, **getmouse**, **ungetmouse**, **mousemask**, **wenclose**, **mouse\_trafo**, **wmouse\_trafo**, **mouseinterval** – mouse interface through curses

**SYNOPSIS**

```
#include <curses.h>

typedef unsigned long mmask_t;

typedef struct {
    short id;      /* ID to distinguish multiple devices */
    int x, y, z;   /* event coordinates */
    mmask_t bstate; /* button state bits */
} MEVENT;

bool has_mouse(void);
int getmouse(MEVENT *event);
int ungetmouse(MEVENT *event);
mmask_t mousemask(mmask_t newmask, mmask_t *oldmask);
bool wenclose(const WINDOW *win, int y, int x);
bool mouse_trafo(int* pY, int* pX, bool to_screen);
bool wmouse_trafo(const WINDOW* win, int* pY, int* pX,
                  bool to_screen);
int mouseinterval(int erval);
```

**DESCRIPTION**

These functions provide an interface to mouse events from **ncurses**(3NCURSES). Mouse events are represented by **KEY\_MOUSE** pseudo-key values in the **wgetch** input stream.

**mousemask**

To make mouse events visible, use the **mousemask** function. This will set the mouse events to be reported. By default, no mouse events are reported. The function will return a mask to indicate which of the specified mouse events can be reported; on complete failure it returns 0. If oldmask is non-NULL, this function fills the indicated location with the previous value of the given window's mouse event mask.

As a side effect, setting a zero mousemask may turn off the mouse pointer; setting a nonzero mask may turn it on. Whether this happens is device-dependent.

**Mouse events**

Here are the mouse event type masks which may be defined:

<i>Name</i>	<i>Description</i>
BUTTON1_PRESSED	mouse button 1 down
BUTTON1_RELEASED	mouse button 1 up
BUTTON1_CLICKED	mouse button 1 clicked
BUTTON1_DOUBLE_CLICKED	mouse button 1 double clicked
BUTTON1_TRIPLE_CLICKED	mouse button 1 triple clicked
BUTTON2_PRESSED	mouse button 2 down
BUTTON2_RELEASED	mouse button 2 up
BUTTON2_CLICKED	mouse button 2 clicked
BUTTON2_DOUBLE_CLICKED	mouse button 2 double clicked
BUTTON2_TRIPLE_CLICKED	mouse button 2 triple clicked
BUTTON3_PRESSED	mouse button 3 down
BUTTON3_RELEASED	mouse button 3 up
BUTTON3_CLICKED	mouse button 3 clicked
BUTTON3_DOUBLE_CLICKED	mouse button 3 double clicked
BUTTON3_TRIPLE_CLICKED	mouse button 3 triple clicked
BUTTON4_PRESSED	mouse button 4 down
BUTTON4_RELEASED	mouse button 4 up
BUTTON4_CLICKED	mouse button 4 clicked
BUTTON4_DOUBLE_CLICKED	mouse button 4 double clicked



mouse(3NCURSES)

mouse(3NCURSES)

BUTTON4_TRIPLE_CLICKED	mouse button 4 triple clicked
BUTTON5_PRESSED	mouse button 5 down
BUTTON5_RELEASED	mouse button 5 up
BUTTON5_CLICKED	mouse button 5 clicked
BUTTON5_DOUBLE_CLICKED	mouse button 5 double clicked
BUTTON5_TRIPLE_CLICKED	mouse button 5 triple clicked
BUTTON_SHIFT	shift was down during button state change
BUTTON_CTRL	control was down during button state change
BUTTON_ALT	alt was down during button state change
ALL_MOUSE_EVENTS	report all button state changes
REPORT_MOUSE_POSITION	report mouse movement

**getmouse**

Once a class of mouse events has been made visible in a window, calling the **wgetch** function on that window may return **KEY\_MOUSE** as an indicator that a mouse event has been queued. To read the event data and pop the event off the queue, call **getmouse**. This function will return **OK** if a mouse event is actually visible in the given window, **ERR** otherwise. When **getmouse** returns **OK**, the data deposited as *y* and *x* in the event structure coordinates will be screen-relative character-cell coordinates. The returned state mask will have exactly one bit set to indicate the event type. The corresponding data in the queue is marked invalid. A subsequent call to **getmouse** will retrieve the next older item from the queue.

**ungetmouse**

The **ungetmouse** function behaves analogously to **ungetch**. It pushes a **KEY\_MOUSE** event onto the input queue, and associates with that event the given state data and screen-relative character-cell coordinates.

**wenclose**

The **wenclose** function tests whether a given pair of screen-relative character-cell coordinates is enclosed by a given window, returning **TRUE** if it is and **FALSE** otherwise. It is useful for determining what subset of the screen windows enclose the location of a mouse event.

**wmouse\_trafo**

The **wmouse\_trafo** function transforms a given pair of coordinates from stdscr-relative coordinates to coordinates relative to the given window or vice versa. The resulting stdscr-relative coordinates are not always identical to window-relative coordinates due to the mechanism to reserve lines on top or bottom of the screen for other purposes (see the **ripoffline** and **slk\_init** calls, for example).

- If the parameter **to\_screen** is **TRUE**, the pointers **pY**, **pX** must reference the coordinates of a location inside the window **win**. They are converted to window-relative coordinates and returned through the pointers. If the conversion was successful, the function returns **TRUE**.
- If one of the parameters was **NULL** or the location is not inside the window, **FALSE** is returned.
- If **to\_screen** is **FALSE**, the pointers **pY**, **pX** must reference window-relative coordinates. They are converted to stdscr-relative coordinates if the window **win** encloses this point. In this case the function returns **TRUE**.
- If one of the parameters is **NULL** or the point is not inside the window, **FALSE** is returned. The referenced coordinates are only replaced by the converted coordinates if the transformation was successful.

**mouse\_trafo**

The **mouse\_trafo** function performs the same translation as **wmouse\_trafo**, using stdscr for **win**.

**mouseinterval**

The **mouseinterval** function sets the maximum time (in thousands of a second) that can elapse between press and release events for them to be recognized as a click. Use **mouseinterval(0)** to disable click resolution. This function returns the previous interval value. Use **mouseinterval(-1)** to obtain the interval without altering it. The default is one sixth of a second.

**has\_mouse**

The **has\_mouse** function returns **TRUE** if the mouse driver has been successfully initialized.

Note that mouse events will be ignored when input is in cooked mode, and will cause an error beep



mouse(3NCURSES)

mouse(3NCURSES)

when cooked mode is being simulated in a window by a function such as **getstr** that expects a linefeed for input-loop termination.

## RETURN VALUE

**getmouse** and **ungetmouse** return the integer **ERR** upon failure or **OK** upon successful completion:

### **getmouse**

returns an error.

- If no mouse driver was initialized, or if the mask parameter is zero,
- It also returns an error if no more events remain in the queue.

### **ungetmouse**

returns an error if the FIFO is full.

**mousemask** returns the mask of reportable events.

**mouseinterval** returns the previous interval value, unless the terminal was not initialized. In that case, it returns the maximum interval value (166).

**wenclose** and **wmouse\_trafo** are boolean functions returning **TRUE** or **FALSE** depending on their test result.

## PORTABILITY

These calls were designed for **ncurses(3NCURSES)**, and are not found in SVr4 curses, 4.4BSD curses, or any other previous version of curses.

The feature macro **NCURSES\_MOUSE\_VERSION** is provided so the preprocessor can be used to test whether these features are present. If the interface is changed, the value of **NCURSES\_MOUSE\_VERSION** will be incremented. These values for **NCURSES\_MOUSE\_VERSION** may be specified when configuring ncurses:

- 1 has definitions for reserved events. The mask uses 28 bits.
- 2 adds definitions for button 5, removes the definitions for reserved events. The mask uses 29 bits.

The order of the **MEVENT** structure members is not guaranteed. Additional fields may be added to the structure in the future.

Under **ncurses(3NCURSES)**, these calls are implemented using either xterm's built-in mouse-tracking API or platform-specific drivers including

- Alessandro Rubini's gpm server
- FreeBSD sysmouse
- OS/2 EMX

If you are using an unsupported configuration, mouse events will not be visible to **ncurses(3NCURSES)** (and the **mousemask** function will always return 0).

If the terminfo entry contains a **XM** string, this is used in the xterm mouse driver to control the way the terminal is initialized for mouse operation. The default, if **XM** is not found, corresponds to private mode 1000 of xterm:

```
\E[?1000%?%p1%{1}%=%th%el%;
```

The **z** member in the event structure is not presently used. It is intended for use with touch screens (which may be pressure-sensitive) or with 3D-mice/trackballs/power gloves.

The **ALL\_MOUSE\_EVENTS** class does not include **REPORT\_MOUSE\_POSITION**. They are distinct. For example, in xterm, wheel/scrolling mice send position reports as a sequence of presses of buttons 4 or 5 without matching button-releases.

## BUGS

Mouse events under xterm will not in fact be ignored during cooked mode, if they have been enabled by **mousemask**. Instead, the xterm mouse report sequence will appear in the string read.

Mouse events under xterm will not be detected correctly in a window with its keypad bit off, since they are interpreted as a variety of function key. Your terminfo description should have **kmous** set to **"\E[M"** (the beginning of the response from xterm for mouse clicks). Other values for **kmous** are per-



mouse(3NCURSES)

mouse(3NCURSES)

mitted, but under the same assumption, i.e., it is the beginning of the response.

Because there are no standard terminal responses that would serve to identify terminals which support the xterm mouse protocol, **ncurses** assumes that if your `$TERM` environment variable contains "xterm", or **kmous** is defined in the terminal description, then the terminal may send mouse events.

**SEE ALSO**

**ncurses**(3NCURSES), **kernel**(3NCURSES), **slk**(3NCURSES), **curses\_variables**(3NCURSES).

