NAME

has_mouse, **getmouse**, **ungetmouse**, **mousemask**, **wenclose**, **mouse_trafo**, **wmouse_trafo**, **mouseinterval**, **mmask_t**, **MEVENT** − get mouse events in *ncurses*

SYNOPSIS

```
#include <curses.h>

/* data types */
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;

/* functions */
bool has_mouse(void);

mmask_t mousemask(mmask_t new-mask, mmask_t * old-mask);

int getmouse(MEVENT * event);
int ungetmouse(MEVENT * event);

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

ncurses provides an interface to the mouse or other pointing device. An application can register its interest in such events; the library then exposes the availability of a mouse event via an *input character reading function*: this is **wgetch**(3X) in the non-wide character *curses* API and **wget_wch**(3X) in the wide character API. A queue distinct from that used for keyboard events accumulates the details of mouse events. The input character reading function reports the **KEY_MOUSE** key code when a mouse event is available for collection. A single mouse event queue serves all windows associated with the screen.

The *MEVENT* structure describes a mouse event. Its*y* and *x* coordinates are screen-, not windo w-, relative. The*bstate* member has e xactly one bit set indicating the event type.

ncurses ignores mouse events when input is in canonical (“cooked”) mode, and produces an error beep when they occur while the library simulates canonical mode in a window, as with **getstr**(3X) (wide-character API users: **get_wstr**(3X)), which expects a line feed to terminate its input loop.

has_mouse

The terminal type or operating system interface must support the encoding of mouse events. **has_mouse** returns **TRUE** if *ncurses*’s mouse driver initialized successfully, and **FALSE** otherwise.

mousemask

Use **mousemask** to select the varieties of mouse event your application wishes to receive. By default, *ncurses* reports no mouse events.

- The function returns an updated copy of *new-mask* indicating which event types of interest are reportable by the terminal’s mouse protocol.

  If the screen is not initialized, or the terminal interface does not report mouse events, **mousemask** returns 0.

- If *old-mask* is not a null pointer, **mousemask** stores the previous value of the screen’s mouse event mask there.

As a side effect, setting a zero mouse mask may turn off the mouse cursor; setting a nonzero mask may turn

it on. Whether this happens is device-dependent.

Mouse Events

Several mouse event types may be selected; construct a mask by logically "or"-ing their values.

Name	Description
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
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	a shift key was down during button state change
BUTTON_CTRL	a control key was down during button state change
BUTTON_ALT	an alt key was down during button state change
ALL_MOUSE_EVENTS	report all button state changes
REPORT_MOUSE_POSITION	report mouse movement

getmouse

When a window is configured to report a non-empty set of event types, calling the input character reading function on that window may return **KEY_MOUSE** to indicate availability of an enqueued mouse event. To read the event data and remove it from the queue, call **getmouse**, which returns **OK** if a mouse event is visible in the given window and **ERR** otherwise. When **getmouse** returns **OK**, it deposits data describing the mouse event in the *event* pointer you supply. A subsequent**getmouse** call retrie ves the next older event from the queue.

ungetmouse

ungetmouse behaves analogously to **ungetch**(3X). It pushes a**KEY_MOUSE** e vent onto the screen's input queue, and *event* onto the mouse event queue.

wenclose

wenclose returns **TRUE** if the pair of screen-relative coordinates (*y*, *x*) is enclosed by the given window *win*, and **FALSE** otherwise. If *win* is a pad, **wenclose** uses its most recent screen coordinates as specified in a **prefresh**(3X) or **pnoutrefresh**(3X) call.

wenclose is useful for determining what subset of the screen's windows encloses the location of a mouse event; it is otherwise independent of the *ncurses* mouse API.

wmouse_trafo

wmouse_trafo transforms the given pair of coordinate pointers (*pY*, *pX*) from a *win*-relative basis to a screen-relative one or *vice versa*, as *to-screen* is **TRUE** or **FALSE**, respectively. **stdscr**-relative coordinates are not always identical to screen coordinates: *curses* supports reservation of screen lines at the top and/or bottom for other purposes; see **ripoffline**(3X) and **slk_init**(3X).

If *to-screen is* **TRUE** and the pointers (*pY*, *pX*) reference coordinates inside *win*, *ncurses* updates their values to **stdscr**-relative coordinates and returns **TRUE**. If either*pY* or *pX* is a null pointer , or (*pY*, *pX*) is not inside *win*, **wmouse_trafo** returns **FALSE**.

If *to-screen is* **FALSE** and the pointers (*pY*, *pX*) reference coordinates inside **stdscr**, *ncurses* updates their values to *win*-relative coordinates and returns **TRUE**. If either*pY* or *pX* is a null pointer , or (*pY*, *pX*) is not inside **stdscr**, **wmouse_trafo** returns **FALSE**.

mouse_trafo

mouse_trafo applies the **wmouse_trafo** translation to **stdscr**. If no screen lines are reserved by **ripoffline**(3X) or **slk_init**(3X), this is the identity transformation.

mouseinterval

mouseinterval sets the maximum time (in thousandths of a second) that can elapse between press and release events for them to be resolved as a *click*. An application might interpret button press and release events separated by more than the mouse interval as a “long press”, or, with motion, as a “drag”.

When *ncurses* detects a mouse event, it awaits further input activity up to this interval, and then checks for a subsequent mouse event which can be combined with the first event. If the timeout expires without input activity, then no click resolution occurs. Calling **mouseinterval(0)** disables click resolution.

mouseinterval returns the previous interval value. Use**mouseinter val(−1)** to obtain the interval without altering it.

The mouse interval is set to one sixth of a second when the corresponding screen is initialized, e.g., in **initscr**(3X) or **setupterm**(3X).

RETURN VALUE

has_mouse, **wenclose**, **mouse_trafo**, and **wmouse_trafo** return **TRUE** or **FALSE** as noted above.

getmouse and **ungetmouse** return **ERR** upon failure and **OK** upon success.

getmouse fails if:

- no mouse driver was initialized,
- the mask of reportable events is zero,
- a mouse event was detected that does not match the mask, or
- no more events remain in the queue.

ungetmouse returns **ERR** if the event queue 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).

NOTES

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

Under *ncurses*, 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 are not visible to *ncurses* (and the **mousemask** function always returns **0**).

If the terminal type possesses the (nonstandard) *terminfo* string capability **XM**, *ncurses*'s *xterm* mouse driver uses it when initializing the terminal for mouse operation. The default, if **XM** is not found, corresponds to private mode 1000 of *xterm*.

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

ncurses also recognizes *xterm*'s newer private mode 1006.

```
\E[?1006;1000%?%p1%{1}%=%th%el%;
```

The *id* member of the mouse event structure is not presently used; no terminal type or operating system interface supports reporting events from distinguishable pointing devices. If you synthesize an*MEVENT*, use an *id* of 0.

The *z* member of the mouse 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.

EXTENSIONS

These functions are *ncurses* extensions, and are not found in SVr4 *curses*, 4.4BSD *curses*, or any other previous *curses* implementation. (SVr4 *curses* did have a *getmouse* function, which took no argument and returned an *unsigned long*.)

PORTABILITY

Applications employing the *ncurses* mouse extension should condition its use on the visibility of the **NCURSES_MOUSE_VERSION** preprocessor macro. When the interface changes, the macro's value increments. Multiple versions are available when *ncurses* is configured; see section "ALTERNATE CONFIGURATIONS" of**ncurses**(3X). The follo wing values may be specified.

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.

HISTORY

SVr4 (1989) added mouse support to its variant of *xterm*(1). It is mentioned in a few places, with little supporting documentation.

- Its "libcurses" manual page lists functions for this feature prototyped in *curses.h*.

```
extern int mouse_set(long int);
extern int mouse_on(long int);
extern int mouse_off(long int);
extern int request_mouse_pos(void);
extern int map_button(unsigned long);
extern void wmouse_position(WINDOW *, int *, int *);
extern unsigned long getmouse(void), getbmap(void);
```

- Its "terminfo" manual page lists capabilities for the feature.

buttons	**btns**	**BT**	Number of buttons on the mouse
get_mouse	**getm**	**Gm**	Curses should get button events
key_mouse	**kmous**	**Km**	0631, Mouse event has occurred
mouse_info	**minfo**	**Mi**	Mouse status information
req_mouse_pos	**reqmp**	**RQ**	Request mouse position report

- The interface made assumptions (as does *ncurses*) about the escape sequences sent to and received from the terminal.

  For instance, the SVr4 *curses* library used the **get_mouse** (**getm**) capability to tell the terminal which mouse button events it should send, passing the mouse-button bit mask to the terminal. Also, it could ask the terminal where the mouse was using the **req_mouse_pos** (**reqmp**) capability.

  Those features required a terminal program that had been modified to work with SVr4 *curses*. They were not part of the X Consortium's *xterm*.

When developing the *xterm* mouse support for *ncurses* in September 1995, Eric Raymond was uninterested in using the same interface due to its lack of documentation. Later, in 1998, Mark Hesseling provided support in *PDCurses* 2.3 using the SVr4 interface. *PDCurses*, however, does not use video terminals, making it unnecessary to be concerned about compatibility with the escape sequences.

BUGS

Mouse events from *xterm* are *not* ignored in canonical ("cooked") mode if they have been enabled by **mousemask**. Instead, the*xterm* mouse report sequence appears in the string read.

An *ncurses* window must enable **keypad**(3X) to correctly receive mouse event reports from *xterm* since they are encoded like function keys. Set the terminal's *terminfo* capability **key_mouse** (**kmous**) to "\E[M" (the beginning of the response from *xterm* for mouse clicks). Other values of **key_mouse** are permitted under the same assumption — that is, a mouse report begins with the value of the **key_mouse** (**kmous**) string capability.

Because there are no standard response sequences that serve to identify terminals supporting the *xterm* mouse protocol, *ncurses* assumes that if **key_mouse** (**kmous**) is defined in the terminal description, or if the terminal type's primary name or aliases contain the string "xterm", then the terminal may send mouse events. *ncurses* checks the **kmous** cap-code first, allowing use of newer *xterm* mouse protocols, such as its private mode 1006.

SEE ALSO

curses(3X), **curs_inopts**(3X), **curs_kernel**(3X), **curs_pad**(3X), **curs_slk**(3X), **curs_variables**(3X)