NAME

getstr, **getnstr**, **wgetstr**, **wgetnstr**, **mvgetstr**, **mvgetnstr**, **mvwgetstr**, **mvwgetnstr** - accept character strings from *curses* terminal keyboard

SYNOPSIS

```
#include <curses.h>

int getstr(char *str);
int getnstr(char *str, int n);
int wgetstr(WINDOW *win, char *str);
int wgetnstr(WINDOW *win, char *str, int n);

int mvgetstr(int y, int x, char *str);
int mvwgetstr(WINDOW *win, int y, int x, char *str);
int mvgetnstr(int y, int x, char *str, int n);
int mvwgetnstr(WINDOW *win, int y, int x, char *str, int n);
```

DESCRIPTION

The function **wgetnstr** is equivalent to a series of calls to **wgetch**(3X), until a newline or carriage return terminates the series:

- The terminating character is not included in the returned string.
- In all instances, the end of the string is terminated by a NUL.
- The function stores the result in the area pointed to by the *str* parameter.
- The function reads at most *n* characters, thus preventing a possible overflow of the input buffer.

  Any attempt to enter more characters (other than the terminating newline or carriage return) causes a beep.

  Function keys also cause a beep and are ignored.

The user's *erase* and *kill* characters are interpreted:

- The *erase* character (e.g., ^**H**) erases the character at the end of the buffer, moving the cursor to the left.

  If *keypad* mode is on for the window, **KEY_LEFT** and **KEY_BACKSPACE** are both considered

equivalent to the user's *erase* character.

- The *kill* character (e.g., **^U**) erases the entire buffer, leaving the cursor at the beginning of the buffer.

Characters input are echoed only if **echo** is currently on. In that case, backspace is echoed as deletion of the previous character (typically a left motion).

The **getnstr**, **mvgetnstr**, **mvwgetnstr**, and **wgetnstr** functions are identical to the **getstr**, **mvgetstr**, **mvwgetstr**, and **wgetstr** functions, respectively, except that the ***n*** versions read at most *n* characters, letting the application prevent overflow of the input buffer.

RETURN VALUE

All of these functions return the integer **OK** upon successful completion. (SVr4 specifies only "an integer value other than **ERR**") If unsuccessful, they return **ERR**.

X/Open defines no error conditions.

In this implementation, these functions return an error

- if the window pointer is null,
- if its timeout expires without having any data, or
- if the associated call to **wgetch** failed.

This implementation provides an extension as well. If a **SIGWINCH** interrupts the function, it will return **KEY_RESIZE** rather than **OK** or **ERR**.

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

NOTES

Any of these functions other than **wgetnstr** may be macros.

Using **getstr**, **mvgetstr**, **mvwgetstr**, or **wgetstr** to read a line that overflows the array pointed to by **str** causes undefined results. The use of **getnstr**, **mvgetnstr**, **mvwgetnstr**, or **wgetnstr**, respectively, is recommended.

PORTABILITY

These functions are described in The Single Unix Specification, Version 2. No error conditions are defined.

This implementation returns **ERR** if the window pointer is null, or if the lower-level **wgetch**(3X) call returns an **ERR**.

SVr3 and early SVr4 curses implementations did not reject function keys; the SVr4.0 documentation claimed that "special keys" (such as function keys, "home" key, "clear" key, *etc*.) are "interpreted", without giving details. It lied. In fact, the "character" value appended to the string by those implementations was predictable but not useful (being, in fact, the low-order eight bits of the key's KEY_ value).

The functions **getnstr**, **mvgetnstr**, and **mvwgetnstr** were present but not documented in SVr4.

X/Open Curses, Issue 5 (2007) stated that these functions "read at most *n* bytes" but did not state whether the terminating NUL is counted in that limit. X/Open Curses, Issue 7 (2009) changed that to say they "read at most *n*-1 bytes" to allow for the terminating NUL. As of 2018, some implementations count it, some do not:

- *ncurses* 6.1 and PDCurses do not count the NUL in the given limit, while
- Solaris SVr4 and NetBSD curses count the NUL as part of the limit.
- Solaris xcurses provides both: its wide-character **wget_nstr** reserves a NUL, but its **wgetnstr** does not count the NUL consistently.

In SVr4 curses, a negative value of *n* tells **wgetnstr** to assume that the caller's buffer is large enough to hold the result, i.e., to act like **wgetstr**. X/Open Curses does not mention this (or anything related to negative or zero values of *n*), however most implementations use the feature, with different limits:

- Solaris SVr4 curses and PDCurses limit the result to 255 bytes. Other Unix systems than Solaris are likely to use the same limit.
- Solaris xcurses limits the result to **LINE_MAX** bytes.
- NetBSD 7 assumes no particular limit for the result from **wgetstr**. However, it limits the **wgetnstr** parameter *n* to ensure that it is greater than zero.

  A comment in NetBSD's source code states that this is specified in SUSv2.

- *ncurses* (before 6.2) assumes no particular limit for the result from **wgetstr**, and treats the *n* parameter of **wgetnstr** like SVr4 curses.

- *ncurses* 6.2 uses **LINE_MAX**, or a larger (system-dependent) value which the **sysconf** function may provide. If neither **LINE_MAX** or **sysconf** is available, *ncurses* uses the POSIX value for **LINE_MAX** (a 2048 byte limit). In either case, it reserves a byte for the terminating NUL.

Although **getnstr** is equivalent to a series of calls to **getch**, it also makes changes to the curses modes to allow simple editing of the input buffer:

- **getnstr** saves the current value of the **nl**, **echo**, **raw** and **cbreak** modes, and sets **nl**, **noecho**, **noraw**, and **cbreak**.

  getnstr handles the echoing of characters, rather than relying on the caller to set an appropriate mode.

- It also obtains the *erase* and *kill* characters from **erasechar** and **killchar**, respectively.

- On return, **getnstr** restores the modes to their previous values.

Other implementations differ in their treatment of special characters:

- While they may set the *echo* mode, other implementations do not modify the *raw* mode, They may take the *cbreak* mode set by the caller into account when deciding whether to handle echoing within **getnstr** or as a side-effect of the **getch** calls.

- The original *ncurses* (as *pcurses* in 1986) set **noraw** and **cbreak** when accepting input for **getnstr**. That may have been done to make function- and cursor-keys work; it is not necessary with *ncurses*.

  Since 1995, *ncurses* has provided signal handlers for INTR and QUIT (e.g., **^C** or ^\). With the **noraw** and **cbreak** settings, those may catch a signal and stop the program, where other implementations allow one to enter those characters in the buffer.

- Starting in 2021 (*ncurses* 6.3), **getnstr** sets **raw**, rather than **noraw** and **cbreak** for better compatibility with SVr4-curses, e.g., allowing one to enter a **^C** into the buffer.

SEE ALSO

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

curses(3X), **curs_getch**(3X), **curs_termattrs**(3X), **curs_variables**(3X)