NAME

del_curterm, **mvcur**, **putp**, **restartterm**, **set_curterm**, **setupterm**, **tigetflag**, **tigetnum**, **tigetstr**, **tiparm**, **tiparm_s**, **tiscan_s**, **tparm**, **tputs**, **vid_attr**, **vid_puts**, **vidattr**, **vidputs** - *curses* interfaces to *terminfo* database

SYNOPSIS

```
#include <curses.h>
#include <term.h>

TERMINAL *cur_term;

const char * const boolnames[];
const char * const boolcodes[];
const char * const boolfnames[];
const char * const numnames[];
const char * const numcodes[];
const char * const numfnames[];
const char * const strnames[];
const char * const strcodes[];
const char * const strfnames[];

int setupterm(const char *term, int filedes, int *errret);
TERMINAL *set_curterm(TERMINAL *nterm);
int del_curterm(TERMINAL *oterm);
int restartterm(const char *term, int filedes, int *errret);

char *tparm(const char *str, ...);
        /* or */
char *tparm(const char *str, long p1 ... long p9);

int tputs(const char *str, int affcnt, int (*putc)(int));
int putp(const char *str);

int vidputs(chtype attrs, int (*putc)(int));
int vidattr(chtype attrs);
int vid_puts(attr_t attrs, short pair, void *opts, int (*putc)(int));
int vid_attr(attr_t attrs, short pair, void *opts);

int mvcur(int oldrow, int oldcol, int newrow, int newcol);
```

int tigetflag(const char ****cap-code*);**
int tigetnum(const char ****cap-code*);**
char *tigetstr(const char ****cap-code*);**

char *tiparm(const char ****str*, ...);**

/ extensions */*
char *tiparm_s(int *expected***, int** *mask***, const char ****str***, ...);**
int tiscan_s(int ****expected*, int ****mask***, const char ****str***);**

/ deprecated */*
int setterm(const char ****term*);**

DESCRIPTION

These low-level functions must be called by programs that deal directly with the *terminfo* database to handle certain terminal capabilities, such as programming function keys. For all other functionality, *curses* functions are more suitable and their use is recommended.

None of these functions use (or are aware of) multibyte character strings such as UTF-8.

- Capability names and codes use the POSIX portable character set.

- Capability string values have no associated encoding; they are strings of 8-bit characters.

Initialization

Initially, **setupterm** should be called. The high-level *curses* functions **initscr** and **newterm** call **setupterm** to initialize the low-level set of terminal-dependent variables listed in **term_variables**(3X).

Applications can use the terminal capabilities either directly (via header definitions), or by special functions. The header files *curses.h* and *term.h* should be included (in that order) to get the definitions for these strings, numbers, and flags.

The *terminfo* variables **lines** and **columns** are initialized by **setupterm** as follows.

- If **use_env(FALSE)** has been called, values for **lines** and **columns** specified in *terminfo* are used.

- Otherwise, if the environment variables *LINES* and *COLUMNS* exist, their values are used. If these environment variables do not exist and the program is running in a window, the current window size is used. Otherwise, if the environment variables do not exist, the values for **lines** and **columns** specified in the *terminfo* database are used.

Parameterized strings should be passed through **tparm** to instantiate them. All *terminfo* strings (including the output of **tparm**) should be sent to the terminal device with **tputs** or **putp**. Call **reset_shell_mode** to restore the terminal modes before exiting; see **curs_kernel**(3X).

Programs that use cursor addressing should

- output **enter_ca_mode** upon startup and
- output **exit_ca_mode** before exiting.

Programs that execute shell subprocesses should

- call **reset_shell_mode** and output **exit_ca_mode** before the shell is called and
- output **enter_ca_mode** and call **reset_prog_mode** after returning from the shell.

setupterm reads in the *terminfo* database, initializing the *terminfo* structures, but does not set up the output virtualization structures used by *curses*. Its parameters follow.

term is the terminal type, a character string. If *term* is null, the environment variable *TERM* is read.

filedes
is the file descriptor used for getting and setting terminal I/O modes.

Higher-level applications use **newterm**(3X) to initialize the terminal, passing an output *stream* rather than a *descriptor*. In *curses*, the two are the same because **newterm** calls **setupterm**, passing the file descriptor derived from its output stream parameter.

errret points to an optional location where an error status can be returned to the caller. If *errret* is not null, then **setupterm** returns **OK** or **ERR** and stores a status value in the integer pointed to by *errret*. A return value of **OK** combined with status of **1** in *errret* is normal.

If **ERR** is returned, examine *errret:*

1 means that the terminal is hardcopy, and cannot be used for *curses* applications.

setupterm determines if the entry is a hardcopy type by checking the **hardcopy** (**hc**) capability.

0 means that the terminal could not be found, or that it is a generic type, having too little information for *curses* applications to run.

setupterm determines if the entry is a generic type by checking the **generic_type** (**gn**) capability.

-1 means that the *terminfo* database could not be found.

If *errret* is null, **setupterm** reports an error message upon finding an error and exits. Thus, the simplest call is:

```
setupterm((char *)0, 1, (int *)0);
```

which uses all the defaults and sends the output to **stdout**.

The Terminal State

setupterm stores its information about the terminal in a *TERMINAL* structure pointed to by the global variable **cur_term**. If it detects an error, or decides that the terminal is unsuitable (hardcopy or generic), it discards this information, making it not available to applications.

If **setupterm** is called repeatedly for the same terminal type, it will reuse the information. It maintains only one copy of a given terminal's capabilities in memory. If it is called for different terminal types, **setupterm** allocates new storage for each set of terminal capabilities.

set_curterm sets **cur_term** to *nterm*, and makes all of the *terminfo* Boolean, numeric, and string variables use the values from *nterm*. It returns the old value of **cur_term**.

del_curterm frees the space pointed to by *oterm* and makes it available for further use. If *oterm* is the same as **cur_term**, references to any of the *terminfo* Boolean, numeric, and string variables thereafter may refer to invalid memory locations until another **setupterm** has been called.

restartterm is similar to **setupterm** and **initscr**, except that it is called after restoring memory to a previous state (for example, when reloading a game saved as a core image dump). **restartterm** assumes that the windows and the input and output options are the same as when memory was saved, but the terminal type and baud rate may be different. Accordingly, **restartterm** saves various terminal state bits, calls **setupterm**, and then restores the bits.

Formatting Output

tparm instantiates the string *str* with parameters *pi*. A pointer is returned to the result of *str* with the parameters applied. Application developers should keep in mind these quirks of the interface:

- Although **tparm**'s actual parameters may be integers or strings, the prototype expects *long* (integer) values.

- Aside from the **set_attributes** (**sgr**) capability, most terminal capabilities require no more than one or two parameters.

- Padding information is ignored by **tparm**; it is interpreted by **tputs**.

- The capability string is null-terminated. Use "\200" where an ASCII NUL is needed in the output.

tiparm is a newer form of **tparm** which uses *stdarg.h* rather than a fixed-parameter list. Its numeric parameters are *int*s rather than *long*s.

Both **tparm** and **tiparm** assume that the application passes parameters consistent with the terminal description. Two extensions are provided as alternatives to deal with untrusted data.

- **tiparm_s** is an extension which is a safer formatting function than **tparm** or **tiparm**, because it allows the developer to tell the *curses* library how many parameters to expect in the parameter list, and which may be string parameters.

  The *mask* parameter has one bit set for each of the parameters (up to 9) passed as *char* pointers rather than numbers.

- The extension **tiscan_s** allows the application to inspect a formatting capability to see what the *curses* library would assume.

Output Functions

String capabilities can contain padding information, a time delay (accommodating performance limitations of hardware terminals) expressed as **$<***n***>**, where *n* is a nonnegative integral count of milliseconds. If *n* exceeds 30,000 (thirty seconds), it is capped at that value.

tputs interprets time-delay information in the string *str* and outputs it, executing the delays:

- The *str* parameter must be a *terminfo* string variable or the return value of **tparm**, **tiparm**, **tgetstr**, or **tgoto**.

  The **tgetstr** and **tgoto** functions are part of the *termcap* interface, which happens to share these function names with the *terminfo* API.

- *affcnt* is the number of lines affected, or **1** if not applicable.

- *putc* is a *putchar*-like function to which the characters are passed, one at a time.

  If **tputs** processes a time-delay, it uses the **delay_output**(3X) function, routing any resulting padding characters through this function.

putp calls "**tputs(***str***, 1, putchar)**". The output of **putp** always goes to **stdout**, rather than the *filedes* specified in **setupterm**.

vidputs displays the string on the terminal in the video attribute mode *attrs*, which is any combination of the attributes listed in **curses**(3X). The characters are passed to the *putchar*-like function *putc*.

vidattr is like **vidputs**, except that it outputs through *putchar*(3).

vid_attr and **vid_puts** correspond to **vidattr** and **vidputs**, respectively. They use multiple parameters to represent the character attributes and color; namely,

- *attrs*, of type *attr_t*, for the attributes and

- *pair*, of type *short*, for the color pair number.

Use the attribute constants prefixed with "**WA_**" with **vid_attr** and **vid_puts**.

X/Open Curses reserves the *opts* argument for future use, saying that applications must provide a null pointer for that argument; but see section "EXTENSIONS" below.

mvcur provides low-level cursor motion. It takes effect immediately (rather than at the next refresh). Unlike the other low-level output functions, which either write to the standard output or pass an output function parameter, **mvcur** uses an output file descriptor derived from the output stream parameter of **newterm**(3X).

While **putp** and **mvcur** are low-level functions that do not use high-level *curses* state, *ncurses* declares them in *curses.h* because System V did this (see section "HISTORY" below).

Terminal Capability Functions

tigetflag, **tigetnum**, and **tigetstr** return the value of the capability corresponding to the *terminfo cap-code*, such as **xenl**, passed to them. The *cap-code* for each capability is given in the table column entitled *cap-code* code in the capabilities section of **terminfo**(5).

These functions return special values to denote errors.

tigetflag returns

-1 if *cap-code* is not a Boolean capability, or

0 if it is canceled or absent from the terminal description.

tigetnum returns

-2 if *cap-code* is not a numeric capability, or

-1 if it is canceled or absent from the terminal description.

tigetstr returns

(char *)-1
if *cap-code* is not a string capability, or

0 if it is canceled or absent from the terminal description.

Terminal Capability Names

These null-terminated arrays contain

- the short *terminfo* names ("codes"),
- the *termcap* names ("names"), and
- the long *terminfo* names ("fnames")

for each of the predefined *terminfo* variables:

const char *boolnames[], ***boolcodes[]**, ***boolfnames[]**
const char *numnames[], ***numcodes[]**, ***numfnames[]**
const char *strnames[], ***strcodes[]**, ***strfnames[]**

Releasing Memory

Each successful call to **setupterm** allocates memory to hold the terminal description. As a side effect, it sets **cur_term** to point to this memory. If an application calls

```
del_curterm(cur_term);
```

the memory will be freed.

The formatting functions **tparm** and **tiparm** extend the storage allocated by **setupterm** as follows.

- They add the "static" *terminfo* variables [a-z]. Before *ncurses* 6.3, those were shared by all screens. With *ncurses* 6.3, those are allocated per screen. See **terminfo**(5).

- To improve performance, *ncurses* 6.3 caches the result of analyzing *terminfo* strings for their parameter types. That is stored as a binary tree referenced from the *TERMINAL* structure.

The higher-level **initscr** and **newterm** functions use **setupterm**. Normally they do not free this memory, but it is possible to do that using the **delscreen**(3X) function.

RETURN VALUE

X/Open Curses defines no failure conditions. In *ncurses*,

del_curtem
: fails if its terminal parameter is null.

putp calls **tputs**, returning the same error codes.

restartterm
: fails if the associated call to **setupterm** returns an error.

setupterm
: fails if it cannot allocate enough memory, or create the initial windows (**stdscr**, **curscr**, and **newscr**) Other error conditions are documented above.

tparm
: returns a null pointer if the capability would require unexpected parameters; that is, too many, too few, or incorrect types (strings where integers are expected, or vice versa).

tputs fails if the string parameter is null. It does not detect I/O errors: X/Open Curses states that **tputs** ignores the return value of the output function *putc*.

NOTES

The **vid_attr** function in *ncurses* is a special case. It was originally implemented based on a draft of X/Open Curses, as a macro, before other parts of the *ncurses* wide-character API were developed, and unlike the other wide-character functions, is also provided in the non-wide-character configuration.

EXTENSIONS

The functions marked as extensions were designed for *ncurses*, and are not found in SVr4 *curses*, 4.4BSD *curses*, or any other previous *curses* implementation.

ncurses allows *opts* to be a pointer to *int*, which overrides the *pair* (*short*) argument.

PORTABILITY

setterm is not described by X/Open and must be considered non-portable. All other functions are as described by X/Open.

Compatibility Macros

This implementation provides a few macros for compatibility with systems before SVr4 (see section "HISTORY" below). They include **Bcrmode**, **Bfixterm**, **Bgettmode**, **Bnocrmode**, **Bresetterm**, **Bsaveterm**, and **Bsetterm**.

In SVr4, these are found in *curses.h*, but except for **setterm**, are likewise macros. The one function, **setterm**, is mentioned in the manual page. It further notes that **setterm** was replaced by **setupterm**, stating that the call

setupterm(*term*, 1, (int *)0)

provides the same functionality as **setterm(***term***)**, discouraging the latter for new programs. *ncurses* implements each of these symbols as macros for BSD *curses* compatibility.

Legacy Data

setupterm copies the terminal name to the array **ttytype**. This is not part of X/Open Curses, but is assumed by some applications.

Other implementions may not declare the capability name arrays. Some provide them without declaring them. X/Open Curses does not specify them.

Extended terminal capability names, as defined by "**tic -x**", are not stored in the arrays described here.

Output Buffering

Older versions of *ncurses* assumed that the file descriptor passed to **setupterm** from **initscr** or **newterm** uses buffered I/O, and would write to the corresponding stream. In addition to the limitation that the terminal was left in block-buffered mode on exit (like System V *curses*), it was problematic because *ncurses* did not allow a reliable way to clean up on receiving **SIGTSTP**.

The current version (ncurses6) uses output buffers managed directly by *ncurses*. Some of the low-level functions described in this manual page write to the standard output. They are not signal-safe. The high-level functions in *ncurses* employ alternate versions of these functions using the more reliable

buffering scheme.

Function Prototypes

The X/Open Curses prototypes are based on the SVr4 *curses* header declarations, which were defined at the same time the C language was first standardized in the late 1980s.

- X/Open Curses uses *const* less effectively than a later design might, sometimes applying it needlessly to values that are already constant, and in most cases overlooking parameters that normally would use *const*. Passing *const*-qualified parameters to functions that do not declare them *const* may prevent the program from compiling. On the other hand, "writable strings" are an obsolescent feature.

  As an extension, this implementation can be configured to change the function prototypes to use the *const* keyword. The *ncurses* ABI 6 enables this feature by default.

- X/Open Curses prototypes **tparm** with a fixed number of parameters, rather than a variable argument list.

  This implementation uses a variable argument list, but can be configured to use the fixed-parameter list. Portable applications should provide nine parameters after the format; zeroes are fine for this purpose.

  In response to review comments by Thomas E. Dickey, X/Open Curses Issue 7 proposed the **tiparm** function in mid-2009.

  While **tiparm** is always provided in *ncurses*, the older form is only available as a build-time configuration option. If not specially configured, **tparm** is the same as **tiparm**.

Both forms of **tparm** have drawbacks:

- Most of the calls to **tparm** use only one or two parameters. Passing nine on each call is awkward.

  Using *long* for the numeric parameter type is a workaround to make the parameter use the same amount of stack as a pointer. That approach dates back to the mid-1980s, before C was standardized. Since then, there is a standard (and pointers are not required to fit in a *long*).

- Providing the right number of parameters for a variadic function such as **tiparm** can be a problem, in particular for string parameters. However, only a few *terminfo* capabilities use string parameters (for instance, the ones used for programmable function keys).

The *ncurses* library checks usage of these capabilities, and returns an error if the capability mishandles string parameters. But it cannot check if a calling program provides strings in the right places for the **tparm** calls.

The **tput**(1) program checks its use of these capabilities with a table, so that it calls **tparm** correctly.

Special *TERM* treatment

If configured to use the terminal driver, as with the MinGW port,

- **setupterm** interprets a missing/empty *TERM* variable as the special value "unknown".

  SVr4 *curses* uses the special value "dumb".

  The difference between the two is that the former uses the **generic_type** (**gn**) *terminfo* capability, while the latter does not. A generic terminal is unsuitable for full-screen applications.

- **setupterm** allows explicit use of the the windows console driver by checking if **$TERM** is set to "#win32con" or an abbreviation of that string.

Other Portability Issues

In SVr4, **set_curterm** returns an *int*, **OK** or **ERR**. We have chosen to implement the X/Open Curses semantics.

In SVr4, the third argument of **tputs** has the type "**int (*putc)(char)**".

At least one implementation of X/Open Curses (Solaris) returns a value other than **OK** or **ERR** from **tputs**. It instead returns the length of the string, and does no error checking.

X/Open Curses notes that after calling **mvcur**, the *curses* state may not match the actual terminal state, and that an application should touch and refresh the window before resuming normal *curses* calls. Both *ncurses* and SVr4 *curses* implement **mvcur** using the *SCREEN* data allocated in either **initscr** or **newterm**. So though it is documented as a *terminfo* function, **mvcur** is really a *curses* function that is not well specified.

X/Open Curses states that the old location must be given for **mvcur** to accommodate terminals that lack absolute cursor positioning. *ncurses* allows the caller to use -1 for either or both old coordinates. The -1 tells *ncurses* that the old location is unknown, and that it must use only absolute motion, as with the **cursor_address** (**cup**) capability, rather than the least costly combination of absolute and relative motion.

HISTORY

SVr2 (1984) introduced the *terminfo* feature. Its programming manual mentioned the following low-level functions.

Function	Description
fixterm	restore terminal to "in *curses*" state
gettmode	establish current terminal modes
mvcur	low level cursor motion
putp	use **tputs** to send characters via *putchar*
resetterm	set terminal modes to "out of *curses*" state
resetty	reset terminal flags to stored value
saveterm	save current modes as "in *curses*" state
savetty	store current terminal flags
setterm	establish terminal with given type
setupterm	establish terminal with given type
tparm	interpolate parameters into string capability
tputs	apply padding information to a string
vidattr	like **vidputs**, but output through *putchar*
vidputs	write string to terminal, applying specified attributes

The programming manual also mentioned functions provided for *termcap* compatibility (commenting that they "may go away at a later date").

FunctionDescription

tgetent look up *termcap* entry for given *name*
tgetflag get Boolean entry for given *id*
tgetnum get numeric entry for given *id*
tgetstr get string entry for given *id*
tgoto apply parameters to given capability
tputs write characters via a function parameter, applying padding

Early *terminfo* programs obtained capability values from the *TERMINAL* structure initialized by **setupterm**.

SVr3 (1987) extended *terminfo* by adding functions to retrieve capability values (like the *termcap* interface), and reusing **tgoto** and **tputs**.

FunctionDescription

---

tigetflag get Boolean entry for given *id*
tigetnum get numeric entry for given *id*
tigetstr get string entry for given *id*

SVr3 also replaced several of the SVr2 *terminfo* functions that had no counterpart in the *termcap* interface, documenting them as obsolete.

Function Replaced by

---

crmode cbreak
fixterm reset_prog_mode
gettmode *n/a*
nocrmode nocbreak
resetterm reset_shell_mode
saveterm def_prog_mode

setterm setupterm

SVr3 kept the **mvcur**, **vidattr**, and **vidputs** functions, along with **putp**, **tparm**, and **tputs**. The latter were needed to support padding, and to handle capabilities accessed by functions such as **vidattr** (which used more than the two parameters supported by **tgoto**).

SVr3 introduced the functions for switching between terminal descriptions; for example, **set_curterm**. Some changes reflected incremental improvements to the SVr2 library.

- The *TERMINAL* type definition was introduced in SVr3.01, for the *term* structure provided in SVr2.

- Various global variables such as **boolnames** were mentioned in the programming manual at this point, though the variables had been provided in SVr2.

SVr4 (1989) added the **vid_attr** and **vid_puts** functions.

Other low-level functions are declared in the *curses* header files of Unix systems, but none are documented. Those noted as "obsolete" by SVr3 remained in use by System V's *vi*(1) editor.

SEE ALSO

curses(3X), **curs_initscr**(3X), **curs_kernel**(3X), **curs_memleaks**(3X), **curs_termcap**(3X), **curs_variables**(3X), **putc**(3), **term_variables**(3X), **terminfo**(5)