NAME

del_curterm, **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 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 lower-level functions of the *curses* standard 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

Call **setupterm** from your application to have *terminfo* manage the terminal device; this action initializes the terminal-dependent variables listed in **term_variables**(3X). (A*cur ses* application calling **initscr**(3X) or **newterm**(3X) achieves the same result.) Applications can use the terminal capabilities either directly, by object definitions corresponding to capability names and codes (see **term_variables**(3X)) or by calling the functions documented here. **setupterm** initializes the *terminfo* v ariables **lines** and **columns** as described in **use_env**(3X).

Pass parameterized string capability values 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**(3X) to restore the terminal modes before exiting. (A*cur ses* application calling **endwin**(3X) achieves the same result.)

Programs that use cursor addressing should emit certain capabilities at specific times. Specifically, output

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

Programs that execute shell subprocesses should

- call **reset_shell_mode**(3X) and output **exit_ca_mode** before the shell is called, and
- output **enter_ca_mode** and call **reset_prog_mode**(3X) 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*cur ses*, 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*err et* 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 a hard-copy type (lacks destructive backspace), and cannot be used for *curses* applications. The library determines this fact by checking the terminal type's **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 *) NULL, 1, (int *) NULL);
```

  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 unavailable to applications.

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

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

del_curterm releases the memory pointed to by *oterm*. If*oterm* is the same as **cur_term**, references to any of the *terminfo* Boolean, numeric, and string capabilities thereafter may refer to invalid memory locations until **setupterm** is called again.

restartterm is similar to **setupterm**, but is intended for use after restoring program memory to a previous state (for example, when reloading an application that has been suspended from one terminal session and restored in another). **r estartterm** assumes that the display dimensions and the input and output options are the same as when memory was saved, but the terminal type and line speed may differ. Accordingly, **restartterm** saves relevant terminal state, calls **setupterm**, then restores that state.

Formatting Output

tparm instantiates the string *str* with parameters *pi*. It returns a pointer to a character string representing *str* with the parameters applied to "%" expressions within. Application developers should keep in mind some 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** that uses *stdarg.h* rather than a fixed-length parameter list. Its numeric parameters are *int*s rather than *long*s.

tparm and **tiparm** assume that the application passes parameters consistent with the terminal description. *ncurses* provides two extensions as alternatives to deal with untrusted data.

- The **tiparm_s** extension 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*, 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), *ncurses* caps it at that value.

tputs interprets time delays in the string *str* and acts upon them.

- The *str* parameter must be a *terminfo* string capability or the return value of **tparm** or **tiparm**.
- *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.

tputs processes each time delay with the **delay_output**(3X) function, routing any resulting padding characters through this function.

putp calls "**tputs(***str***, 1, putchar)**". The output of**putp** al ways 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*putc har*-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.

While **putp** is a lower-level function that does not use higher-level *curses* state, *ncurses* declares it in *curses.h* because System V did so (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 of that name 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

NULL 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 standard *terminfo* capability name.

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*TERMIN AL* 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

Functions that return integers return **ERR** upon failure and **OK** upon success.

In *ncurses*,

del_curterm
: fails if its terminal parameter is null.

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

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

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 originated in *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

Except for *setterm*, X/Open Curses Issue 4 describes these functions. It specifies no error conditions for them.

SVr4 describes a successful return value except where "otherwise noted" as "an integer value other than *ERR*".

Header Files

On legacy *curses* systems, include *curses.h* and *term.h* in that order to make visible the definitions of the string arrays storing the capability names and codes.

Compatibility Macros

ncurses implements a few macros for early System V *curses* compatibility (see section "HISTORY" below). They include *crmode*, *fixterm*, *gettmode*, *nocrmode*, *resetterm*, *saveterm*, and *setterm*.

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)
```

and discouraging the latter for new programs.

Legacy Data

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

Other implementations 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* used buffered I/O, and wrote to the corresponding *stdio* 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*.

ncurses 6.*x* uses output buffers managed directly by *ncurses*. The lower-level functions described here that write to the terminal device do so via the standard output stream; they thus are not signal-safe. The higher-level functions in *ncurses* employ alternate versions of these functions using a 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 function parameters that are passed by value (and therefore copied), and in most cases overlooking parameters that normally would benefit from *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 C language feature.

  As an extension, *ncurses* 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.

  ncurses 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 available only as a build-time configuration option. If not specially configured,*tparm* is the same as *tiparm*.

Both forms of *tparm* have drawbacks.

- Most calls to *tparm* require 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 memory as a pointer. That approach dates to the mid-1980s, before C was standardized. Since ANSI C (1989), C language standards do not require a pointer 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 *ERR* if the capability mishandles string parameters. But it cannot check if a calling program provides strings in the right places for the *tparm* calls.

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

Special *TERM* treatment

If *ncurses* is configured to use a terminal driver that does not employ the POSIX *termios* API, as with the MinGW port,

- *setupterm* interprets a missing or 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 Microsoft Windows console driver by checking whether the*TERM* environment variable has the value "#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 (*)(char)**".

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

Very old versions of AIX *curses* required inclusion of *curses.h* before *term.h*.

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").

Function	Description
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*.

Function	Description
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)