NAME

curses_trace, **trace**, **_tracef**, **_traceattr**, **_traceattr2**, **_tracecchar_t**, **_tracecchar_t2**, **_tracechar**, **_tracechtype**, **_tracechtype2**, **_nc_tracebits**, **_tracedump**, **_tracemouse - curses** debugging routines

SYNOPSIS

```
#include <curses.h>

unsigned curses_trace(const unsigned param);

void _tracef(const char *format, ...);

char *_traceattr(attr_t attr);
char *_traceattr2(int buffer, chtype ch);
char *_tracecchar_t(const cchar_t *string);
char *_tracecchar_t2(int buffer, const cchar_t *string);
char *_tracechar(int ch);
char *_tracechtype(chtype ch);
char *_tracechtype2(int buffer, chtype ch);

void _tracedump(const char *label, WINDOW *win);
char *_nc_tracebits(void);
char *_tracemouse(const MEVENT *event);

/* deprecated */
void trace(const unsigned int param);
```

DESCRIPTION

The *curses trace* routines are used for debugging the ncurses libraries, as well as applications which use the ncurses libraries. Some limitations apply:

- Aside from **curses_trace**, the other functions are normally available only with the debugging library e.g., *libncurses_g.a*.

  All of the trace functions may be compiled into any model (shared, static, profile) by defining the symbol **TRACE**.

- Additionally, the functions which use **cchar_t** are only available with the wide-character configuration of the libraries.

Functions

The principal parts of this interface are

- **curses_trace**, which selectively enables different tracing features, and
- **_tracef**, which writes formatted data to the *trace* file.

  The other functions either return a pointer to a string-area (allocated by the corresponding function), or return no value (such as **_tracedump**, which implements the screen dump for **TRACE_UPDATE**). The caller should not free these strings, since the allocation is reused on successive calls. To work around the problem of a single string-area per function, some use a buffer-number parameter, telling the library to allocate additional string-areas.

The **curses_trace** function is always available, whether or not the other trace functions are available:

- If tracing is available, calling **curses_trace** with a nonzero parameter updates the trace mask, and returns the previous trace mask.

  When the trace mask is nonzero, ncurses creates the file "trace" in the current directory for output. If the file already exists, no tracing is done.

- If tracing is not available, **curses_trace** returns zero (0).

Trace Parameter

The trace parameter is formed by OR'ing values from the list of **TRACE**_*xxx* definitions in **<curses.h>**. These include:

TRACE_DISABLE
: turn off tracing by passing a zero parameter.

  The library flushes the output file, but retains an open file-descriptor to the trace file so that it can resume tracing later if a nonzero parameter is passed to the **curses_trace** function.

TRACE_TIMES
: trace user and system times of updates.

TRACE_TPUTS
: trace **tputs**(3X) calls.

TRACE_UPDATE
: trace update actions, old & new screens.

TRACE_MOVE
trace cursor movement and scrolling.

TRACE_CHARPUT
trace all character outputs.

TRACE_ORDINARY
trace all update actions. The old and new screen contents are written to the trace file for each refresh.

TRACE_CALLS
trace all curses calls. The parameters for each call are traced, as well as return values.

TRACE_VIRTPUT
trace virtual character puts, i.e., calls to **addch**.

TRACE_IEVENT
trace low-level input processing, including timeouts.

TRACE_BITS
trace state of TTY control bits.

TRACE_ICALLS
trace internal/nested calls.

TRACE_CCALLS
trace per-character calls.

TRACE_DATABASE
trace read/write of terminfo/termcap data.

TRACE_ATTRS
trace changes to video attributes and colors.

TRACE_MAXIMUM
maximum trace level, enables all of the separate trace features.

Some tracing features are enabled whenever the **curses_trace** parameter is nonzero. Some features overlap. The specific names are used as a guideline.

Initialization

These functions check the **NCURSES_TRACE** environment variable, to set the tracing feature as if **curses_trace** was called:

filter, initscr, new_prescr, newterm, nofilter, restartterm, ripoffline, setupterm, slk_init, tgetent, use_env, use_extended_names, use_tioctl

Command-line Utilities

The command-line utilities such as **tic**(1) provide a verbose option which extends the set of messages written using the **curses_trace** function. Both of these (**-v** and **curses_trace**) use the same variable (**_nc_tracing**), which determines the messages which are written.

Because the command-line utilities may call initialization functions such as **setupterm**, **tgetent** or **use_extended_names**, some of their debugging output may be directed to the *trace* file if the **NCURSES_TRACE** environment variable is set:

- messages produced in the utility are written to the standard error.
- messages produced by the underlying library are written to *trace*.

If ncurses is built without tracing, none of the latter are produced, and fewer diagnostics are provided by the command-line utilities.

RETURN VALUE

Routines which return a value are designed to be used as parameters to the **_tracef** routine.

PORTABILITY

These functions are not part of the XSI interface. Some other curses implementations are known to have similar features, but they are not compatible with ncurses:

- SVr4 provided **traceon** and **traceoff**, to control whether debugging information was written to the "trace" file. While the functions were always available, this feature was only enabled if **DEBUG** was defined when building the library.

  The SVr4 tracing feature is undocumented.

- PDCurses provides **traceon** and **traceoff**, which (like SVr4) are always available, and enable tracing to the "trace" file only when a debug-library is built.

  PDCurses has a short description of these functions, with a note that they are not present in

X/Open Curses, ncurses or NetBSD. It does not mention SVr4, but the functions' inclusion in a header file section labeled "Quasi-standard" hints at the origin.

- NetBSD does not provide functions for enabling/disabling traces. It uses environment variables **CURSES_TRACE_MASK** and **CURSES_TRACE_FILE** to determine what is traced, and where the results are written. This is available only when a debug-library is built.

  The NetBSD tracing feature is undocumented.

A few ncurses functions are not provided when symbol versioning is used:

_nc_tracebits, _tracedump, _tracemouse

The original **trace** routine was deprecated because it often conflicted with application names.

SEE ALSO

curses(3X).