NAME

scr_dump, **scr_restore**, **scr_init**, **scr_set** − read/write a *curses* screen from/to a file

SYNOPSIS

#include <curses.h>

int scr_dump(const char * *filename***);**
int scr_restore(const char * *filename***);**
int scr_init(const char * *filename***);**
int scr_set(const char * *filename***);**

DESCRIPTION

curses provides applications the ability to write the contents of the screen to a file and read them back. To read/write a window (rather than the whole screen) from/to a file, use **getwin**(3X) and **putwin**(3X), respectively.

scr_dump

scr_dump writes to *filename* the contents of the virtual screen; see **curscr**(3X).

scr_restore

scr_restore updates the virtual screen to match the contents of *filename* (if validly written with **scr_dump**). *curses* does not perform a refresh; after making any desired changes, call **doupdate**(3X) or similar.

scr_init

scr_init reads *filename*, using it to initialize *curses* data structures describing the state of the terminal screen. *curses* then, if it decides the terminal state is valid, bases its next update of the screen on this information rather than clearing it and starting from scratch.

curses regards the terminal as in an invalid state for computation of updates based on the contents of *filename* if

- *curses* knows that the terminal has been written to since the preceding **scr_dump** call, or
- the terminal type supports the *terminfo* capabilities **exit_ca_mode** (**rmcup**) or **non_rev_rmcup** (**nrrmc**).

Either of the foregoing conditions means that *curses* cannot assume that the terminal's contents match their representation in *filename*. The former is due to terminal features (such as*xterm*(1)' s "alternate screen") that couple cursor-positioning mode with a local cache of screen contents. *curses* cannot know whether the terminal is displaying from that local cache at the time the application calls **scr_init**, so it makes a pessimistic assumption that a full redraw is required; see subsection "Cursor Motions" of **terminfo**(5).

scr_init could be used after **initscr**(3X) or *system*(3) to share the screen with another process that has done a **scr_dump** after **endwin**(3X). An application that supports suspending its state on exit and subsequent resumption upon later execution might use **scr_dump** and **scr_init** thus.

scr_set

scr_set combines **scr_restore** and **scr_init**, synchronizing the contents of *filename* with the virtual screen. It can be regarded as a screen inheritance function; consider a real-time screen-sharing application.

RETURN VALUE

These functions return **OK** on success and **ERR** on failure.

In *ncurses*, each function returns **ERR** if it cannot open *filename*. **scr_init**, **scr_restore**, and **scr_set** return **ERR** if the contents of *filename* are invalid.

NOTES

scr_init, **scr_restore**, and **scr_set** may be implemented as macros.

PORTABILITY

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

SVr4 omitted the *const* qualifiers.

SVr4 documentation describes *scr_init* such that the dump data is also considered invalid "if the time-stamp of the tty is old" but does not define "old".

As of 2024, *PDCurses* provides these functions. NetBSD *curses* does not.

Other implementations of *curses* store the window in binary form, which makes the dump dependent upon the *curses* library's internal data structures. *ncur ses* avoids this drawback by storing the dump in textual form, allowing more flexible use of the data. For instance, the *scr_restore* of SVr4 *curses* requires that the dumped window have the same dimensions as the restored window. *ncurses* uses its **wresize**(3X) extension to adjust the restored window size.

HISTORY

SVr3 (1987) introduced *scr_dump*, *scr_init*, and *scr_restore*. SVr3.1 added*scr_set*.

SEE ALSO

curses(3X), **curs_initscr**(3X), **curs_refresh**(3X), **curs_util**(3X), **system**(3), **scr_dump**(5), **terminfo**(5), **wresize**(3X)