Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'term.5' command

$ man term.5

term(5) File Formats Manual term(5)

NAME

term - format of compiled term file.

SYNOPSIS

term

DESCRIPTION

STORAGE LOCATION

Compiled terminfo descriptions are placed under the directory /usr/share/terminfo. Two configurations are supported (when building the ncurses libraries):

directory tree

A two-level scheme is used to avoid a linear search of a huge UNIX system directory: /usr/share/terminfo/c/name where name is the name of the terminal, and c is the first character of name. Thus, act4 can be found in the file /usr/share/terminfo/a/act4. Syn? onyms for the same terminal are implemented by multiple links to the same compiled file.

hashed database

Using Berkeley database, two types of records are stored: the ter? minfo data in the same format as stored in a directory tree with the terminfo's primary name as a key, and records containing only aliases pointing to the primary name.

If built to write hashed databases, ncurses can still read ter?

minfo databases organized as a directory tree, but cannot write entries into the directory tree. It can write (or rewrite) en? tries in the hashed database.

ncurses distinguishes the two cases in the TERMINFO and TER? MINFO_DIRS environment variable by assuming a directory tree for entries that correspond to an existing directory, and hashed data? base otherwise.

LEGACY STORAGE FORMAT

The format has been chosen so that it will be the same on all hardware. An 8 or more bit byte is assumed, but no assumptions about byte order? ing or sign extension are made.

The compiled file is created with the tic program, and read by the rou? tine setupterm(3X). The file is divided into six parts:

a) header,

b) terminal names,

c) boolean flags,

d) numbers,

e) strings, and

f) string table.

The header section begins the file. This section contains six short integers in the format described below. These integers are

(1) the magic number (octal 0432);

(2) the size, in bytes, of the terminal names section;

(3) the number of bytes in the boolean flags section;

(4) the number of short integers in the numbers section;

(5) the number of offsets (short integers) in the strings section;

(6) the size, in bytes, of the string table.

The capabilities in the boolean flags, numbers, and strings sections are in the same order as the file <term.h>.

Short integers are signed, in the range -32768 to 32767. They are stored as two 8-bit bytes. The first byte contains the least signifi? cant 8 bits of the value, and the second byte contains the most signif? icant 8 bits. (Thus, the value represented is 256*second+first.) This format corresponds to the hardware of the VAX and PDP-11 (that is, lit? tle-endian machines). Machines where this does not correspond to the hardware must read the integers as two bytes and compute the little-en? dian value.

Numbers in a terminal description, whether they are entries in the num? bers or strings table, are positive integers. Boolean flags are treated as positive one-byte integers. In each case, those positive integers represent a terminal capability. The terminal compiler tic uses negative integers to handle the cases where a capability is not available:

? If a capability is absent from this terminal, tic stores a -1 in the corresponding table.

The integer value -1 is represented by two bytes 0377, 0377.

Absent boolean values are represented by the byte 0 (false).

? If a capability has been canceled from this terminal, tic stores a -2 in the corresponding table.

The integer value -2 is represented by two bytes 0377, 0376.

The boolean value -2 is represented by the byte 0376.

? Other negative values are illegal.

The terminal names section comes after the header. It contains the first line of the terminfo description, listing the various names for the terminal, separated by the ?|? character. The terminal names sec? tion is terminated with an ASCII NUL character.

The boolean flags section has one byte for each flag. Boolean capabil? ities are either 1 or 0 (true or false) according to whether the termi? nal supports the given capability or not.

Between the boolean flags section and the number section, a null byte will be inserted, if necessary, to ensure that the number section be? gins on an even byte This is a relic of the PDP-11's word-addressed ar? chitecture, originally designed to avoid traps induced by addressing a word on an odd byte boundary. All short integers are aligned on a short word boundary.

The numbers section is similar to the boolean flags section. Each ca?

pability takes up two bytes, and is stored as a little-endian short in? teger.

The strings section is also similar. Each capability is stored as a short integer. The capability value is an index into the string table.

The string table is the last section. It contains all of the values of string capabilities referenced in the strings section. Each string is null-terminated. Special characters in ^X or \c notation are stored in their interpreted form, not the printing representation. Padding in? formation $<nn> and parameter information %x are stored intact in unin? terpreted form.

EXTENDED STORAGE FORMAT

The previous section describes the conventional terminfo binary format. With some minor variations of the offsets (see PORTABILITY), the same binary format is used in all modern UNIX systems. Each system uses a predefined set of boolean, number or string capabilities.

The ncurses libraries and applications support extended terminfo binary format, allowing users to define capabilities which are loaded at run? time. This extension is made possible by using the fact that the other implementations stop reading the terminfo data when they have reached the end of the size given in the header. ncurses checks the size, and if it exceeds that due to the predefined data, continues to parse ac? cording to its own scheme.

First, it reads the extended header (5 short integers):

(1) count of extended boolean capabilities

(2) count of extended numeric capabilities

(3) count of extended string capabilities

(4) count of the items in extended string table

(5) size of the extended string table in bytes

The count- and size-values for the extended string table include the extended capability names as well as extended capability values.

Using the counts and sizes, ncurses allocates arrays and reads data for the extended capabilities in the same order as the header information.

The extended string table contains values for string capabilities. Af?

ter the end of these values, it contains the names for each of the ex?
tended capabilities in order, e.g., booleans, then numbers and finally strings.

Applications which manipulate terminal data can use the definitions de? scribed in term_variables(3X) which associate the long capability names with members of a TERMTYPE structure.

EXTENDED NUMBER FORMAT

On occasion, 16-bit signed integers are not large enough. With ncurses 6.1, a new format was introduced by making a few changes to the legacy format:

? a different magic number (octal 01036)

? changing the type for the number array from signed 16-bit integers to signed 32-bit integers.

To maintain compatibility, the library presents the same data struc? tures to direct users of the TERMTYPE structure as in previous formats. However, that cannot provide callers with the extended numbers. The library uses a similar but hidden data structure TERMTYPE2 to provide data for the terminfo functions.

PORTABILITY

setupterm

Note that it is possible for setupterm to expect a different set of ca? pabilities than are actually present in the file. Either the database may have been updated since setupterm has been recompiled (resulting in extra unrecognized entries in the file) or the program may have been recompiled more recently than the database was updated (resulting in missing entries). The routine setupterm must be prepared for both pos? sibilities - this is why the numbers and sizes are included. Also, new capabilities must always be added at the end of the lists of boolean, number, and string capabilities.

Binary format

X/Open Curses does not specify a format for the terminfo database. UNIX System V curses used a directory-tree of binary files, one per terminal description.

Despite the consistent use of little-endian for numbers and the other? wise self-describing format, it is not wise to count on portability of binary terminfo entries between commercial UNIX versions. The problem is that there are at least three versions of terminfo (under HP-UX, AIX, and OSF/1) which diverged from System V terminfo after SVr1, and have added extension capabilities to the string table that (in the bi? nary format) collide with System V and XSI Curses extensions. See ter? minfo(5) for detailed discussion of terminfo source compatibility is? sues.

This implementation is by default compatible with the binary terminfo format used by Solaris curses, except in a few less-used details where it was found that the latter did not match X/Open Curses. The format used by the other Unix versions can be matched by building ncurses with different configuration options.

Magic codes

The magic number in a binary terminfo file is the first 16-bits (two bytes). Besides making it more reliable for the library to check that a file is terminfo, utilities such as file also use that to tell what the file-format is. System V defined more than one magic number, with 0433, 0435 as screen-dumps (see scr_dump(5)). This implementation uses 01036 as a continuation of that sequence, but with a different high-or? der byte to avoid confusion.

The TERMTYPE structure

Direct access to the TERMTYPE structure is provided for legacy applica? tions. Portable applications should use the tigetflag and related functions described in curs_terminfo(3X) for reading terminal capabili? ties.

Mixed-case terminal names

A small number of terminal descriptions use uppercase characters in their names. If the underlying filesystem ignores the difference be? tween uppercase and lowercase, ncurses represents the ?first character? of the terminal name used as the intermediate level of a directory tree in (two-character) hexadecimal form.

EXAMPLE

As an example, here is a description for the Lear-Siegler ADM-3, a pop?

ular though rather stupid early terminal:

```
adm3a|lsi adm3a,
        am,
        cols#80, lines#24,
        bel=^G, clear= 32$<1>, cr=^M, cub1=^H, cud1=^J,
        cuf1=^L, cup=\E=%p1%{32}%+%c%p2%{32}%+%c, cuu1=^K,
        home=^^, ind=^J,
```

and a hexadecimal dump of the compiled terminal description:

```
0000  1a 01 10 00 02 00 03 00  82 00 31 00 61 64 6d 33  ........ ..1.adm3
0010  61 7c 6c 73 69 20 61 64  6d 33 61 00 00 01 50 00  a|lsi ad m3a...P.
0020  ff ff 18 00 ff ff 00 00  02 00 ff ff ff ff 04 00  ........ ........
0030  ff ff ff ff ff ff ff ff  0a 00 25 00 27 00 ff ff  ........ ..%.'...
0040  29 00 ff ff ff ff 2b 00  ff ff 2d 00 ff ff ff ff  ).....+. ..-.....
0050  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  ........ ........
0060  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  ........ ........
0070  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  ........ ........
0080  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  ........ ........
0090  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  ........ ........
00a0  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  ........ ........
00b0  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  ........ ........
00c0  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  ........ ........
00d0  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  ........ ........
00e0  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  ........ ........
00f0  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  ........ ........
0100  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  ........ ........
0110  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  ........ ........
0120  ff ff ff ff ff ff 2f 00  07 00 0d 00 1a 24 3c 31  ....../. .....$<1
0130  3e 00 1b 3d 25 70 31 25  7b 33 32 7d 25 2b 25 63  >..=%p1% {32}%+%c
0140  25 70 32 25 7b 33 32 7d  25 2b 25 63 00 0a 00 1e  %p2%{32} %+%c....
0150  00 08 00 0c 00 0b 00 0a  00                       ........ .
```

LIMITS

Some limitations:

? total compiled entries cannot exceed 4096 bytes in the legacy for?
mat.

? total compiled entries cannot exceed 32768 bytes in the extended
format.

? the name field cannot exceed 128 bytes.

Compiled entries are limited to 32768 bytes because offsets into the strings table use two-byte integers. The legacy format could have sup? ported 32768-byte entries, but was limited a virtual memory page's 4096 bytes.

FILES

/usr/share/terminfo/*/* compiled terminal capability data base

SEE ALSO

curses(3X), terminfo(5).

AUTHORS

Thomas E. Dickey

extended terminfo format for ncurses 5.0

hashed database support for ncurses 5.6

extended number support for ncurses 6.1

Eric S. Raymond

documented legacy terminfo format, e.g., from pcurses.