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

$ man user_caps.5

user_caps(5)   File Formats Manual   user_caps(5)

NAME

    user_caps - user-defined terminfo capabilities

SYNOPSIS

    tic -x, infocmp -x

DESCRIPTION

  Background

    Before ncurses 5.0, terminfo databases used a fixed repertoire of ter?

    minal capabilities designed for the SVr2 terminal database in 1984, and

    extended in stages through SVr4 (1989), and standardized in the Single

    Unix Specification beginning in 1995.

    Most of the extensions in this fixed repertoire were additions to the

    tables of boolean, numeric and string capabilities. Rather than change

    the meaning of an existing capability, a new name was added. The ter?

    minfo database uses a binary format; binary compatibility was ensured

    by using a header which gave the number of items in the tables for each

    type of capability. The standardization was incomplete:

    ?   The binary format itself is not described in the X/Open Curses doc?

        umentation. Only the source format is described.

        Library developers rely upon the SVr4 documentation, and reverse-

        engineering the compiled terminfo files to match the binary format.

    ?   Lacking a standard for the binary format, most implementations copy

        the SVr2 binary format, which uses 16-bit signed integers, and is

limited to 4096-byte entries.

The format cannot represent very large numeric capabilities, nor can it represent large numbers of special keyboard definitions.

? The tables of capability names differ between implementations. Although they may provide all of the standard capability names, the position in the tables differs because some features were added as needed, while others were added (out of order) to comply with X/Open Curses.

While ncurses' repertoire of predefined capabilities is closest to Solaris, Solaris's terminfo database has a few differences from the list published by X/Open Curses. For example, ncurses can be con? figured with tables which match the terminal databases for AIX, HP-UX or OSF/1, rather than the default Solaris-like configuration.

? In SVr4 curses and ncurses, the terminal database is defined at compile-time using a text file which lists the different terminal capabilities.

In principle, the text-file can be extended, but doing this re? quires recompiling and reinstalling the library. The text-file used in ncurses for terminal capabilities includes details for var? ious systems past the documented X/Open Curses features. For exam? ple, ncurses supports these capabilities in each configuration:

memory_lock

(meml) lock memory above cursor

memory_unlock

(memu) unlock memory

box_chars_1

(box1) box characters primary set

The memory lock/unlock capabilities were included because they were used in the X11R6 terminal description for xterm. The box1 capa? bility is used in tic to help with terminal descriptions written for AIX.

During the 1990s, some users were reluctant to use terminfo in spite of its performance advantages over termcap:

- ? The fixed repertoire prevented users from adding features for unan?
  ticipated terminal improvements (or required them to reuse existing
  capabilities as a workaround).
- ? The limitation to 16-bit signed integers was also mentioned. Be?
  cause termcap stores everything as a string, it could represent
  larger numbers.

Although termcap's extensibility was rarely used (it was never the
speaker who had actually used the feature), the criticism had a point.
ncurses 5.0 provided a way to detect nonstandard capabilities, deter?
mine their type and optionally store and retrieve them in a way which
did not interfere with other applications. These are referred to as
user-defined capabilities because no modifications to the toolset's
predefined capability names are needed.

The ncurses utilities tic and infocmp have a command-line option ?-x?
to control whether the nonstandard capabilities are stored or re?
trieved. A library function use_extended_names is provided for the
same purpose.

When compiling a terminal database, if ?-x? is set, tic will store a
user-defined capability if the capability name is not one of the prede?
fined names.

Because ncurses provides a termcap library interface, these user-de?
fined capabilities may be visible to termcap applications:

- ? The termcap interface (like all implementations of termcap) re?
  quires that the capability names are 2-characters.
  When the capability is simple enough for use in a termcap applica?
  tion, it is provided as a 2-character name.
- ? There are other user-defined capabilities which refer to features
  not usable in termcap, e.g., parameterized strings that use more
  than two parameters or use more than the trivial expression support
  provided by termcap. For these, the terminfo database should have
  only capability names with 3 or more characters.
- ? Some terminals can send distinct strings for special keys (cursor-,
  keypad- or function-keys) depending on modifier keys (shift, con?

trol, etc.).   While terminfo and termcap have a set of 60 prede?

fined function-key names, to which a series of keys can be as?

signed, that is insufficient for more than a dozen keys multiplied

by more than a couple of modifier combinations. The ncurses data?

base uses a convention based on xterm to provide extended special-

key names.

Fitting that into termcap's limitation of 2-character names would

be pointless.   These extended keys are available only with ter?

minfo.

Recognized capabilities

The ncurses library uses the user-definable capabilities.   While the

terminfo database may have other extensions, ncurses makes explicit

checks for these:

AX boolean, asserts that the terminal interprets SGR 39 and SGR 49

by resetting the foreground and background color, respectively,

to the default.

This is a feature recognized by the screen program as well.

E3 string, tells how to clear the terminal's scrollback buffer.

When present, the clear(1) program sends this before clearing the

terminal.

The command ?tput clear? does the same thing.

RGB

boolean, number or string, to assert that the set_a_foreground

and set_a_background capabilities correspond to direct colors,

using an RGB (red/green/blue) convention. This capability allows

the color_content function to return appropriate values without

requiring the application to initialize colors using init_color.

The capability type determines the values which ncurses sees:

boolean

implies that the number of bits for red, green and blue are

the same.   Using the maximum number of colors, ncurses adds

two, divides that sum by three, and assigns the result to red,

green and blue in that order.

If the number of bits needed for the number of colors is not a
multiple of three, the blue (and green) components lose in
comparison to red.

number

tells ncurses what result to add to red, green and blue. If
ncurses runs out of bits, blue (and green) lose just as in the
boolean case.

string

explicitly list the number of bits used for red, green and
blue components as a slash-separated list of decimal integers.

Because there are several RGB encodings in use, applications
which make assumptions about the number of bits per color are un?
likely to work reliably. As a trivial case, for example, one
could define RGB#1 to represent the standard eight ANSI colors,
i.e., one bit per color.

U8 number, asserts that ncurses must use Unicode values for line-
drawing characters, and that it should ignore the alternate char?
acter set capabilities when the locale uses UTF-8 encoding. For
more information, see the discussion of NCURSES_NO_UTF8_ACS in
ncurses(3X).

Set this capability to a nonzero value to enable it.

XM string, override ncurses's built-in string which enables/disables
xterm mouse mode.

ncurses sends a character sequence to the terminal to initialize
mouse mode, and when the user clicks the mouse buttons or (in
certain modes) moves the mouse, handles the characters sent back
by the terminal to tell it what was done with the mouse.

The mouse protocol is enabled when the mask passed in the mouse?
mask function is nonzero. By default, ncurses handles the re?
sponses for the X11 xterm mouse protocol. It also knows about
the SGR 1006 xterm mouse protocol, but must to be told to look
for this specifically. It will not be able to guess which mode
is used, because the responses are enough alike that only confu?

sion would result.

The XM capability has a single parameter. If nonzero, the mouse protocol should be enabled. If zero, the mouse protocol should be disabled. ncurses inspects this capability if it is present, to see whether the 1006 protocol is used. If so, it expects the responses to use the SGR 1006 xterm mouse protocol.

The xterm mouse protocol is used by other terminal emulators. The terminal database uses building-blocks for the various xterm mouse protocols which can be used in customized terminal descrip? tions.

The terminal database building blocks for this mouse feature also have an experimental capability xm. The ?xm? capability de? scribes the mouse response. Currently there is no interpreter which would use this information to make the mouse support com? pletely data-driven.

xm shows the format of the mouse responses. In this experimental capability, the parameters are

  p1   y-ordinate

  p2   x-ordinate

  p3   button

  p4   state, e.g., pressed or released

  p5   y-ordinate starting region

  p6   x-ordinate starting region

  p7   y-ordinate ending region

  p8   x-ordinate ending region

Here are examples from the terminal database for the most com? monly used xterm mouse protocols:

  xterm+x11mouse|X11 xterm mouse protocol,

        kmous=\E[M, XM=\E[?1000%?%p1%{1}%=%th%el%;,

        xm=\E[M

          %?%p4%t%p3%e%{3}%;%' '%+%c

          %p2%'!'%+%c

          %p1%'!'%+%c,

```
xterm+sm+1006|xterm SGR-mouse,

       kmous=\E[<, XM=\E[?1006;1000%?%p1%{1}%=%th%el%;,

       xm=\E[<%i%p3%d;

         %p1%d;

         %p2%d;

         %?%p4%tM%em%;,
```

Extended key-definitions

Several terminals provide the ability to send distinct strings for com?

binations of modified special keys. There is no standard for what

those keys can send.

Since 1999, xterm has supported shift, control, alt, and meta modifiers

which produce distinct special-key strings. In a terminal description,

ncurses has no special knowledge of the modifiers used. Applications

can use the naming convention established for xterm to find these spe?

cial keys in the terminal description.

Starting with the curses convention that key names begin with ?k? and

that shifted special keys are an uppercase name, ncurses' terminal

database defines these names to which a suffix is added:

Name   Description

???

kDC   special form of kdch1 (delete character)

kDN   special form of kcud1 (cursor down)

kEND   special form of kend (End)

kHOM   special form of khome (Home)

kLFT   special form of kcub1 (cursor-left or cursor-back)

kNXT   special form of knext (Next, or Page-Down)

kPRV   special form of kprev (Prev, or Page-Up)

kRIT   special form of kcuf1 (cursor-right, or cursor-forward)

kUP   special form of kcuu1 (cursor-up)

These are the suffixes used to denote the modifiers:

Value   Description

????????????????????????????????????

2   Shift

3   Alt

    4   Shift + Alt

    5   Control

    6   Shift + Control

    7   Alt + Control

    8   Shift + Alt + Control

    9   Meta

    10   Meta + Shift

    11   Meta + Alt

    12   Meta + Alt + Shift

    13   Meta + Ctrl

    14   Meta + Ctrl + Shift

    15   Meta + Ctrl + Alt

    16   Meta + Ctrl + Alt + Shift

None of these are predefined; terminal descriptions can refer to names

which ncurses will allocate at runtime to key-codes. To use these keys

in an ncurses program, an application could do this:

?   using a list of extended key names, ask tigetstr(3X) for their val?

    ues, and

?   given the list of values, ask key_defined(3X) for the key-code

    which would be returned for those keys by wgetch(3X).

PORTABILITY

    The ?-x? extension feature of tic and infocmp has been adopted in Net?

    BSD curses. That implementation stores user-defined capabilities, but

    makes no use of these capabilities itself.

SEE ALSO

    infocmp(1M), tic(1M).

    The terminal database section NCURSES USER-DEFINABLE CAPABILITIES sum?

    marizes commonly-used user-defined capabilities which are used in the

    terminal descriptions.   Some of those features are mentioned in

    screen(1) or tmux(1).

    XTerm Control Sequences provides further information on the xterm fea?

    tures which are used in these extended capabilities.

AUTHORS

Thomas E. Dickey

beginning with ncurses 5.0 (1999)

<div align="center">user_caps(5)</div>