Rocky Enterprise Linux 9.2 Manual Pages on command 'terminfo.5'

$ man terminfo.5

terminfo(5) File Formats terminfo(5)

NAME

terminfo - terminal capability data base

SYNOPSIS

/usr/share/terminfo/*/*

DESCRIPTION

Terminfo is a data base describing terminals, used by screen-oriented programs such as nvi(1), lynx(1), mutt(1), and other curses applica? tions, using high-level calls to libraries such as curses(3X). It is also used via low-level calls by non-curses applications which may be screen-oriented (such as clear(1)) or non-screen (such as tabs(1)).

Terminfo describes terminals by giving a set of capabilities which they have, by specifying how to perform screen operations, and by specifying padding requirements and initialization sequences.

This manual describes ncurses version 6.2 (patch 20210508).

Terminfo Entry Syntax

Entries in terminfo consist of a sequence of fields:

? Each field ends with a comma ?,? (embedded commas may be escaped

with a backslash or written as ?\054?).

? White space between fields is ignored.

? The first field in a terminfo entry begins in the first column.

? Newlines and leading whitespace (spaces or tabs) may be used for formatting entries for readability. These are removed from parsed entries.

The infocmp -f and -W options rely on this to format if-then-else expressions, or to enforce maximum line-width. The resulting for? matted terminal description can be read by tic.

? The first field for each terminal gives the names which are known for the terminal, separated by ?|? characters.

The first name given is the most common abbreviation for the termi? nal (its primary name), the last name given should be a long name fully identifying the terminal (see longname(3X)), and all others are treated as synonyms (aliases) for the primary terminal name.

X/Open Curses advises that all names but the last should be in lower case and contain no blanks; the last name may well contain upper case and blanks for readability.

This implementation is not so strict; it allows mixed case in the primary name and aliases. If the last name has no embedded blanks, it allows that to be both an alias and a verbose name (but will warn about this ambiguity).

? Lines beginning with a ?#? in the first column are treated as com? ments.

While comment lines are legal at any point, the output of captoinfo and infotocap (aliases for tic) will move comments so they occur only between entries.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware mak? ing up the terminal should have a root name, thus ?hp2621?. This name should not contain hyphens. Modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and a mode suf? fix. Thus, a vt100 in 132-column mode would be vt100-w. The following

suffixes should be used where possible:

Suffix	Meaning	Example
-nn	Number of lines on the screen	aaa-60
-np	Number of pages of memory	c100-4p
-am	With automargins (usually the default)	vt100-am
-m	Mono mode; suppress color	ansi-m
-mc	Magic cookie; spaces when highlighting	wy30-mc
-na	No arrow keys (leave them in local)	c100-na
-nam	Without automatic margins	vt100-nam
-nl	No status line	att4415-nl
-ns	No status line	hp2626-ns
-rv	Reverse video	c100-rv
-s	Enable status line	vt100-s
-vb	Use visible bell instead of beep	wy370-vb
-w	Wide mode (> 80 columns, usually 132)	vt100-w

For more on terminal naming conventions, see the term(7) manual page.

Terminfo Capabilities Syntax

The terminfo entry consists of several capabilities, i.e., features that the terminal has, or methods for exercising the terminal's fea? tures.

After the first field (giving the name(s) of the terminal entry), there should be one or more capability fields. These are boolean, numeric or string names with corresponding values:

? Boolean capabilities are true when present, false when absent. There is no explicit value for boolean capabilities.

? Numeric capabilities have a ?#? following the name, then an un? signed decimal integer value.

? String capabilities have a ?=? following the name, then an string of characters making up the capability value.

String capabilities can be split into multiple lines, just as the fields comprising a terminal entry can be split into multiple lines. While blanks between fields are ignored, blanks embedded within a string value are retained, except for leading blanks on a

line.

Any capability can be canceled, i.e., suppressed from the terminal en? try, by following its name with ?@? rather than a capability value.

Similar Terminals

If there are two very similar terminals, one (the variant) can be de? fined as being just like the other (the base) with certain exceptions. In the definition of the variant, the string capability use can be given with the name of the base terminal:

? The capabilities given before use override those in the base type named by use.

? If there are multiple use capabilities, they are merged in reverse order. That is, the rightmost use reference is processed first, then the one to its left, and so forth.

? Capabilities given explicitly in the entry override those brought in by use references.

A capability can be canceled by placing xx@ to the left of the use ref? erence that imports it, where xx is the capability. For example, the entry

2621-nl, smkx@, rmkx@, use=2621,

defines a 2621-nl that does not have the smkx or rmkx capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

An entry included via use can contain canceled capabilities, which have the same effect as if those cancels were inline in the using terminal entry.

Predefined Capabilities

The following is a complete table of the capabilities included in a terminfo description block and available to terminfo-using code. In each line of the table,

The variable is the name by which the programmer (at the terminfo level) accesses the capability.

The capname is the short name used in the text of the database, and is

used by a person updating the database. Whenever possible, capnames are chosen to be the same as or similar to the ANSI X3.64-1979 standard (now superseded by ECMA-48, which uses identical or very similar names). Semantics are also intended to match those of the specifica? tion.

The termcap code is the old termcap capability name (some capabilities are new, and have names which termcap did not originate).

Capability names have no hard length limit, but an informal limit of 5 characters has been adopted to keep them short and to allow the tabs in the source file Caps to line up nicely.

Finally, the description field attempts to convey the semantics of the capability. You may find some codes in the description field:

(P) indicates that padding may be specified

#[1-9] in the description field indicates that the string is passed through tparm(3X) with parameters as given (#i).

If no parameters are listed in the description, passing the string through tparm(3X) may give unexpected results, e.g., if it contains percent (%%) signs.

(P*) indicates that padding may vary in proportion to the number of lines affected

(#i) indicates the ith parameter.

These are the boolean capabilities:

Variable Booleans	Cap-name	TCap Code	Description
auto_left_margin	bw	bw	cub1 wraps from col? umn 0 to last column
auto_right_margin	am	am	terminal has auto? matic margins
back_color_erase	bce	ut	screen erased with background color
can_change	ccc	cc	terminal can re-de? fine existing colors
ceol_standout_glitch	xhp	xs	standout not erased

			by overwriting (hp)
col_addr_glitch	xhpa	YA	only positive motion for hpa/mhpa caps
cpi_changes_res	cpix	YF	changing character pitch changes reso? lution
cr_cancels_micro_mode	crxm	YB	using cr turns off micro mode
dest_tabs_magic_smso	xt	xt	tabs destructive, magic so char (t1061)
eat_newline_glitch	xenl	xn	newline ignored af? ter 80 cols (con? cept)
erase_overstrike	eo	eo	can erase over? strikes with a blank
generic_type	gn	gn	generic line type
hard_copy	hc	hc	hardcopy terminal
hard_cursor	chts	HC	cursor is hard to see
has_meta_key	km	km	Has a meta key (i.e., sets 8th-bit)
has_print_wheel	daisy	YC	printer needs opera? tor to change char? acter set
has_status_line	hs	hs	has extra status line
hue_lightness_saturation	hls	hl	terminal uses only HLS color notation (Tektronix)
insert_null_glitch	in	in	insert mode distin? guishes nulls
lpi_changes_res	lpix	YG	changing line pitch

changes resolution

memory_above da da display may be re?
tained above the
screen

memory_below db db display may be re?
tained below the
screen

move_insert_mode mir mi safe to move while
in insert mode

move_standout_mode msgr ms safe to move while
in standout mode

needs_xon_xoff nxon nx padding will not
work, xon/xoff re?
quired

no_esc_ctlc xsb xb beehive (f1=escape,
f2=ctrl C)

no_pad_char npc NP pad character does
not exist

non_dest_scroll_region ndscr ND scrolling region is
non-destructive

non_rev_rmcup nrrmc NR smcup does not re?
verse rmcup

over_strike os os terminal can over?
strike

prtr_silent mc5i 5i printer will not
echo on screen

row_addr_glitch xvpa YD only positive motion
for vpa/mvpa caps

semi_auto_right_margin sam YE printing in last
column causes cr

status_line_esc_ok eslok es escape can be used
on the status line

tilde_glitch hz hz cannot print ~'s

			(Hazeltine)
transparent_underline	ul	ul	underline character overstrikes
xon_xoff	xon	xo	terminal uses xon/xoff handshaking

These are the numeric capabilities:

Variable Numeric	Cap-name	TCap Code	Description
columns	cols	co	number of columns in a line
init_tabs	it	it	tabs initially every # spaces
label_height	lh	lh	rows in each label
label_width	lw	lw	columns in each la? bel
lines	lines	li	number of lines on screen or page
lines_of_memory	lm	lm	lines of memory if > line. 0 means varies
magic_cookie_glitch	xmc	sg	number of blank characters left by smso or rmso
max_attributes	ma	ma	maximum combined at? tributes terminal can handle
max_colors	colors	Co	maximum number of colors on screen
max_pairs	pairs	pa	maximum number of color-pairs on the screen
maximum_windows	wnum	MW	maximum number of definable windows
no_color_video	ncv	NC	video attributes

			that cannot be used with colors
num_labels	nlab	Nl	number of labels on screen
padding_baud_rate	pb	pb	lowest baud rate where padding needed
virtual_terminal	vt	vt	virtual terminal number (CB/unix)
width_status_line	wsl	ws	number of columns in status line

The following numeric capabilities are present in the SVr4.0 term structure, but are not yet documented in the man page. They came in with SVr4's printer support.

Variable Numeric	Cap-name	TCap Code	Description
bit_image_entwining	bitwin	Yo	number of passes for each bit-image row
bit_image_type	bitype	Yp	type of bit-image device
buffer_capacity	bufsz	Ya	numbers of bytes buffered before printing
buttons	btns	BT	number of buttons on mouse
dot_horz_spacing	spinh	Yc	spacing of dots hor? izontally in dots per inch
dot_vert_spacing	spinv	Yb	spacing of pins ver? tically in pins per inch
max_micro_address	maddr	Yd	maximum value in mi? cro_..._address
max_micro_jump	mjump	Ye	maximum value in

			parm_..._micro
micro_col_size	mcs	Yf	character step size when in micro mode
micro_line_size	mls	Yg	line step size when in micro mode
number_of_pins	npins	Yh	numbers of pins in print-head
output_res_char	orc	Yi	horizontal resolu? tion in units per line
output_res_horz_inch	orhi	Yk	horizontal resolu? tion in units per inch
output_res_line	orl	Yj	vertical resolution in units per line
output_res_vert_inch	orvi	Yl	vertical resolution in units per inch
print_rate	cps	Ym	print rate in char? acters per second
wide_char_size	widcs	Yn	character step size when in double wide mode

These are the string capabilities:

Variable String	Cap- name	TCap Code	Description
acs_chars	acsc	ac	graphics charset pairs, based on vt100
back_tab	cbt	bt	back tab (P)
bell	bel	bl	audible signal (bell) (P)
carriage_return	cr	cr	carriage return (P*) (P*)

change_char_pitch	cpi	ZA	Change number of characters per inch to #1
change_line_pitch	lpi	ZB	Change number of lines per inch to #1
change_res_horz	chr	ZC	Change horizontal resolution to #1
change_res_vert	cvr	ZD	Change vertical res?olution to #1
change_scroll_region	csr	cs	change region to line #1 to line #2 (P)
char_padding	rmp	rP	like ip but when in insert mode
clear_all_tabs	tbc	ct	clear all tab stops (P)
clear_margins	mgc	MC	clear right and left soft margins
clear_screen	clear	cl	clear screen and home cursor (P*)
clr_bol	el1	cb	Clear to beginning of line
clr_eol	el	ce	clear to end of line (P)
clr_eos	ed	cd	clear to end of screen (P*)
column_address	hpa	ch	horizontal position #1, absolute (P)
command_character	cmdch	CC	terminal settable cmd character in prototype !?
create_window	cwin	CW	define a window #1 from #2,#3 to #4,#5

cursor_address cup cm move to row #1 col?
umns #2
cursor_down cud1 do down one line
cursor_home home ho home cursor (if no
cup)
cursor_invisible civis vi make cursor invisi?
ble
cursor_left cub1 le move left one space
cursor_mem_address mrcup CM memory relative cur?
sor addressing, move
to row #1 columns #2
cursor_normal cnorm ve make cursor appear
normal (undo
civis/cvvis)
cursor_right cuf1 nd non-destructive
space (move right
one space)
cursor_to_ll ll ll last line, first
column (if no cup)
cursor_up cuu1 up up one line
cursor_visible cvvis vs make cursor very
visible
define_char defc ZE Define a character
#1, #2 dots wide,
descender #3
delete_character dch1 dc delete character
(P*)
delete_line dl1 dl delete line (P*)
dial_phone dial DI dial number #1
dis_status_line dsl ds disable status line
display_clock dclk DK display clock
down_half_line hd hd half a line down
ena_acs enacs eA enable alternate

			char set
enter_alt_charset_mode	smacs	as	start alternate character set (P)
enter_am_mode	smam	SA	turn on automatic margins
enter_blink_mode	blink	mb	turn on blinking
enter_bold_mode	bold	md	turn on bold (extra bright) mode
enter_ca_mode	smcup	ti	string to start pro? grams using cup
enter_delete_mode	smdc	dm	enter delete mode
enter_dim_mode	dim	mh	turn on half-bright mode
enter_doublewide_mode	swidm	ZF	Enter double-wide mode
enter_draft_quality	sdrfq	ZG	Enter draft-quality mode
enter_insert_mode	smir	im	enter insert mode
enter_italics_mode	sitm	ZH	Enter italic mode
enter_leftward_mode	slm	ZI	Start leftward car? riage motion
enter_micro_mode	smicm	ZJ	Start micro-motion mode
enter_near_letter_quality	snlq	ZK	Enter NLQ mode
enter_normal_quality	snrmq	ZL	Enter normal-quality mode
enter_protected_mode	prot	mp	turn on protected mode
enter_reverse_mode	rev	mr	turn on reverse video mode
enter_secure_mode	invis	mk	turn on blank mode (characters invisi? ble)

enter_shadow_mode	sshm	ZM	Enter shadow-print mode
enter_standout_mode	smso	so	begin standout mode
enter_subscript_mode	ssubm	ZN	Enter subscript mode
enter_superscript_mode	ssupm	ZO	Enter superscript mode
enter_underline_mode	smul	us	begin underline mode
enter_upward_mode	sum	ZP	Start upward car? riage motion
enter_xon_mode	smxon	SX	turn on xon/xoff handshaking
erase_chars	ech	ec	erase #1 characters (P)
exit_alt_charset_mode	rmacs	ae	end alternate char? acter set (P)
exit_am_mode	rmam	RA	turn off automatic margins
exit_attribute_mode	sgr0	me	turn off all at? tributes
exit_ca_mode	rmcup	te	strings to end pro? grams using cup
exit_delete_mode	rmdc	ed	end delete mode
exit_doublewide_mode	rwidm	ZQ	End double-wide mode
exit_insert_mode	rmir	ei	exit insert mode
exit_italics_mode	ritm	ZR	End italic mode
exit_leftward_mode	rlm	ZS	End left-motion mode
exit_micro_mode	rmicm	ZT	End micro-motion mode
exit_shadow_mode	rshm	ZU	End shadow-print mode
exit_standout_mode	rmso	se	exit standout mode
exit_subscript_mode	rsubm	ZV	End subscript mode
exit_superscript_mode	rsupm	ZW	End superscript mode

exit_underline_mode rmul ue exit underline mode

exit_upward_mode rum ZX End reverse charac? ter motion

exit_xon_mode rmxon RX turn off xon/xoff handshaking

fixed_pause pause PA pause for 2-3 sec? onds

flash_hook hook fh flash switch hook

flash_screen flash vb visible bell (may not move cursor)

form_feed ff ff hardcopy terminal page eject (P*)

from_status_line fsl fs return from status line

goto_window wingo WG go to window #1

hangup hup HU hang-up phone

init_1string is1 i1 initialization string

init_2string is2 is initialization string

init_3string is3 i3 initialization string

init_file if if name of initializa? tion file

init_prog iprog iP path name of program for initialization

initialize_color initc Ic initialize color #1 to (#2,#3,#4)

initialize_pair initp Ip Initialize color pair #1 to fg=(#2,#3,#4), bg=(#5,#6,#7)

insert_character ich1 ic insert character (P)

insert_line	il1	al	insert line (P*)
insert_padding	ip	ip	insert padding after inserted character
key_a1	ka1	K1	upper left of keypad
key_a3	ka3	K3	upper right of key? pad
key_b2	kb2	K2	center of keypad
key_backspace	kbs	kb	backspace key
key_beg	kbeg	@1	begin key
key_btab	kcbt	kB	back-tab key
key_c1	kc1	K4	lower left of keypad
key_c3	kc3	K5	lower right of key? pad
key_cancel	kcan	@2	cancel key
key_catab	ktbc	ka	clear-all-tabs key
key_clear	kclr	kC	clear-screen or erase key
key_close	kclo	@3	close key
key_command	kcmd	@4	command key
key_copy	kcpy	@5	copy key
key_create	kcrt	@6	create key
key_ctab	kctab	kt	clear-tab key
key_dc	kdch1	kD	delete-character key
key_dl	kdl1	kL	delete-line key
key_down	kcud1	kd	down-arrow key
key_eic	krmir	kM	sent by rmir or smir in insert mode
key_end	kend	@7	end key
key_enter	kent	@8	enter/send key
key_eol	kel	kE	clear-to-end-of-line key
key_eos	ked	kS	clear-to-end-of-screen key

key_exit	kext	@9	exit key
key_f0	kf0	k0	F0 function key
key_f1	kf1	k1	F1 function key
key_f10	kf10	k;	F10 function key
key_f11	kf11	F1	F11 function key
key_f12	kf12	F2	F12 function key
key_f13	kf13	F3	F13 function key
key_f14	kf14	F4	F14 function key
key_f15	kf15	F5	F15 function key
key_f16	kf16	F6	F16 function key
key_f17	kf17	F7	F17 function key
key_f18	kf18	F8	F18 function key
key_f19	kf19	F9	F19 function key
key_f2	kf2	k2	F2 function key
key_f20	kf20	FA	F20 function key
key_f21	kf21	FB	F21 function key
key_f22	kf22	FC	F22 function key
key_f23	kf23	FD	F23 function key
key_f24	kf24	FE	F24 function key
key_f25	kf25	FF	F25 function key
key_f26	kf26	FG	F26 function key
key_f27	kf27	FH	F27 function key
key_f28	kf28	FI	F28 function key
key_f29	kf29	FJ	F29 function key
key_f3	kf3	k3	F3 function key
key_f30	kf30	FK	F30 function key
key_f31	kf31	FL	F31 function key
key_f32	kf32	FM	F32 function key
key_f33	kf33	FN	F33 function key
key_f34	kf34	FO	F34 function key
key_f35	kf35	FP	F35 function key
key_f36	kf36	FQ	F36 function key
key_f37	kf37	FR	F37 function key

key_f38	kf38	FS	F38 function key
key_f39	kf39	FT	F39 function key
key_f4	kf4	k4	F4 function key
key_f40	kf40	FU	F40 function key
key_f41	kf41	FV	F41 function key
key_f42	kf42	FW	F42 function key
key_f43	kf43	FX	F43 function key
key_f44	kf44	FY	F44 function key
key_f45	kf45	FZ	F45 function key
key_f46	kf46	Fa	F46 function key
key_f47	kf47	Fb	F47 function key
key_f48	kf48	Fc	F48 function key
key_f49	kf49	Fd	F49 function key
key_f5	kf5	k5	F5 function key
key_f50	kf50	Fe	F50 function key
key_f51	kf51	Ff	F51 function key
key_f52	kf52	Fg	F52 function key
key_f53	kf53	Fh	F53 function key
key_f54	kf54	Fi	F54 function key
key_f55	kf55	Fj	F55 function key
key_f56	kf56	Fk	F56 function key
key_f57	kf57	Fl	F57 function key
key_f58	kf58	Fm	F58 function key
key_f59	kf59	Fn	F59 function key
key_f6	kf6	k6	F6 function key
key_f60	kf60	Fo	F60 function key
key_f61	kf61	Fp	F61 function key
key_f62	kf62	Fq	F62 function key
key_f63	kf63	Fr	F63 function key
key_f7	kf7	k7	F7 function key
key_f8	kf8	k8	F8 function key
key_f9	kf9	k9	F9 function key
key_find	kfnd	@0	find key

key_help khlp %1 help key

key_home khome kh home key

key_ic kich1 kI insert-character key

key_il kil1 kA insert-line key

key_left kcub1 kl left-arrow key

key_ll kll kH lower-left key (home down)

key_mark kmrk %2 mark key

key_message kmsg %3 message key

key_move kmov %4 move key

key_next knxt %5 next key

key_npage knp kN next-page key

key_open kopn %6 open key

key_options kopt %7 options key

key_ppage kpp kP previous-page key

key_previous kprv %8 previous key

key_print kprt %9 print key

key_redo krdo %0 redo key

key_reference kref &1 reference key

key_refresh krfr &2 refresh key

key_replace krpl &3 replace key

key_restart krst &4 restart key

key_resume kres &5 resume key

key_right kcuf1 kr right-arrow key

key_save ksav &6 save key

key_sbeg kBEG &9 shifted begin key

key_scancel kCAN &0 shifted cancel key

key_scommand kCMD *1 shifted command key

key_scopy kCPY *2 shifted copy key

key_screate kCRT *3 shifted create key

key_sdc kDC *4 shifted delete-char? acter key

key_sdl kDL *5 shifted delete-line

key

key_select kslt *6 select key

key_send kEND *7 shifted end key

key_seol kEOL *8 shifted clear-to-

end-of-line key

key_sexit kEXT *9 shifted exit key

key_sf kind kF scroll-forward key

key_sfind kFND *0 shifted find key

key_shelp kHLP #1 shifted help key

key_shome kHOM #2 shifted home key

key_sic kIC #3 shifted insert-char?

acter key

key_sleft kLFT #4 shifted left-arrow

key

key_smessage kMSG %a shifted message key

key_smove kMOV %b shifted move key

key_snext kNXT %c shifted next key

key_soptions kOPT %d shifted options key

key_sprevious kPRV %e shifted previous key

key_sprint kPRT %f shifted print key

key_sr kri kR scroll-backward key

key_sredo kRDO %g shifted redo key

key_sreplace kRPL %h shifted replace key

key_sright kRIT %i shifted right-arrow

key

key_srsume kRES %j shifted resume key

key_ssave kSAV !1 shifted save key

key_ssuspend kSPD !2 shifted suspend key

key_stab khts kT set-tab key

key_sundo kUND !3 shifted undo key

key_suspend kspd &7 suspend key

key_undo kund &8 undo key

key_up kcuu1 ku up-arrow key

keypad_local	rmkx	ke	leave 'key? board_transmit' mode
keypad_xmit	smkx	ks	enter 'key? board_transmit' mode
lab_f0	lf0	l0	label on function key f0 if not f0
lab_f1	lf1	l1	label on function key f1 if not f1
lab_f10	lf10	la	label on function key f10 if not f10
lab_f2	lf2	l2	label on function key f2 if not f2
lab_f3	lf3	l3	label on function key f3 if not f3
lab_f4	lf4	l4	label on function key f4 if not f4
lab_f5	lf5	l5	label on function key f5 if not f5
lab_f6	lf6	l6	label on function key f6 if not f6
lab_f7	lf7	l7	label on function key f7 if not f7
lab_f8	lf8	l8	label on function key f8 if not f8
lab_f9	lf9	l9	label on function key f9 if not f9
label_format	fln	Lf	label format
label_off	rmln	LF	turn off soft labels
label_on	smln	LO	turn on soft labels
meta_off	rmm	mo	turn off meta mode
meta_on	smm	mm	turn on meta mode (8th-bit on)
micro_column_address	mhpa	ZY	Like column_address

			in micro mode
micro_down	mcud1	ZZ	Like cursor_down in micro mode
micro_left	mcub1	Za	Like cursor_left in micro mode
micro_right	mcuf1	Zb	Like cursor_right in micro mode
micro_row_address	mvpa	Zc	Like row_address #1 in micro mode
micro_up	mcuu1	Zd	Like cursor_up in micro mode
newline	nel	nw	newline (behave like cr followed by lf)
order_of_pins	porder	Ze	Match software bits to print-head pins
orig_colors	oc	oc	Set all color pairs to the original ones
orig_pair	op	op	Set default pair to its original value
pad_char	pad	pc	padding char (in? stead of null)
parm_dch	dch	DC	delete #1 characters (P*)
parm_delete_line	dl	DL	delete #1 lines (P*)
parm_down_cursor	cud	DO	down #1 lines (P*)
parm_down_micro	mcud	Zf	Like parm_down_cur? sor in micro mode
parm_ich	ich	IC	insert #1 characters (P*)
parm_index	indn	SF	scroll forward #1 lines (P)
parm_insert_line	il	AL	insert #1 lines (P*)
parm_left_cursor	cub	LE	move #1 characters

			to the left (P)
parm_left_micro	mcub	Zg	Like parm_left_cur? sor in micro mode
parm_right_cursor	cuf	RI	move #1 characters to the right (P*)
parm_right_micro	mcuf	Zh	Like parm_right_cur? sor in micro mode
parm_rindex	rin	SR	scroll back #1 lines (P)
parm_up_cursor	cuu	UP	up #1 lines (P*)
parm_up_micro	mcuu	Zi	Like parm_up_cursor in micro mode
pkey_key	pfkey	pk	program function key #1 to type string #2
pkey_local	pfloc	pl	program function key #1 to execute string #2
pkey_xmit	pfx	px	program function key #1 to transmit string #2
plab_norm	pln	pn	program label #1 to show string #2
print_screen	mc0	ps	print contents of screen
prtr_non	mc5p	pO	turn on printer for #1 bytes
prtr_off	mc4	pf	turn off printer
prtr_on	mc5	po	turn on printer
pulse	pulse	PU	select pulse dialing
quick_dial	qdial	QD	dial number #1 with? out checking
remove_clock	rmclk	RC	remove clock
repeat_char	rep	rp	repeat char #1 #2

			times (P*)
req_for_input	rfi	RF	send next input char (for ptys)
reset_1string	rs1	r1	reset string
reset_2string	rs2	r2	reset string
reset_3string	rs3	r3	reset string
reset_file	rf	rf	name of reset file
restore_cursor	rc	rc	restore cursor to position of last save_cursor
row_address	vpa	cv	vertical position #1 absolute (P)
save_cursor	sc	sc	save current cursor position (P)
scroll_forward	ind	sf	scroll text up (P)
scroll_reverse	ri	sr	scroll text down (P)
select_char_set	scs	Zj	Select character set, #1
set_attributes	sgr	sa	define video at? tributes #1-#9 (PG9)
set_background	setb	Sb	Set background color #1
set_bottom_margin	smgb	Zk	Set bottom margin at current line
set_bottom_margin_parm	smgbp	Zl	Set bottom margin at line #1 or (if smgtp is not given) #2 lines from bottom
set_clock	sclk	SC	set clock, #1 hrs #2 mins #3 secs
set_color_pair	scp	sp	Set current color pair to #1
set_foreground	setf	Sf	Set foreground color

			#1
set_left_margin	smgl	ML	set left soft margin
			at current col?
			umn. See smgl.
			(ML is not in BSD
			termcap).
set_left_margin_parm	smglp	Zm	Set left (right)
			margin at column #1
set_right_margin	smgr	MR	set right soft mar?
			gin at current col?
			umn
set_right_margin_parm	smgrp	Zn	Set right margin at
			column #1
set_tab	hts	st	set a tab in every
			row, current columns
set_top_margin	smgt	Zo	Set top margin at
			current line
set_top_margin_parm	smgtp	Zp	Set top (bottom)
			margin at row #1
set_window	wind	wi	current window is
			lines #1-#2 cols
			#3-#4
start_bit_image	sbim	Zq	Start printing bit
			image graphics
start_char_set_def	scsd	Zr	Start character set
			definition #1, with
			#2 characters in the
			set
stop_bit_image	rbim	Zs	Stop printing bit
			image graphics
stop_char_set_def	rcsd	Zt	End definition of
			character set #1
subscript_characters	subcs	Zu	List of subscript?

			able characters
superscript_characters	supcs	Zv	List of superscript? able characters
tab	ht	ta	tab to next 8-space hardware tab stop
these_cause_cr	docr	Zw	Printing any of these characters causes CR
to_status_line	tsl	ts	move to status line, column #1
tone	tone	TO	select touch tone dialing
underline_char	uc	uc	underline char and move past it
up_half_line	hu	hu	half a line up
user0	u0	u0	User string #0
user1	u1	u1	User string #1
user2	u2	u2	User string #2
user3	u3	u3	User string #3
user4	u4	u4	User string #4
user5	u5	u5	User string #5
user6	u6	u6	User string #6
user7	u7	u7	User string #7
user8	u8	u8	User string #8
user9	u9	u9	User string #9
wait_tone	wait	WA	wait for dial-tone
xoff_character	xoffc	XF	XOFF character
xon_character	xonc	XN	XON character
zero_motion	zerom	Zx	No motion for subse? quent character

The following string capabilities are present in the SVr4.0 term struc? ture, but were originally not documented in the man page.

Variable	Cap-	TCap	Description

String	name	Code	
alt_scancode_esc	scesa	S8	Alternate escape for scancode emu? lation
bit_image_carriage_return	bicr	Yv	Move to beginning of same row
bit_image_newline	binel	Zz	Move to next row of the bit image
bit_image_repeat	birep	Xy	Repeat bit image cell #1 #2 times
char_set_names	csnm	Zy	Produce #1'th item from list of char? acter set names
code_set_init	csin	ci	Init sequence for multiple codesets
color_names	colornm	Yw	Give name for color #1
define_bit_image_region	defbi	Yx	Define rectangular bit image region
device_type	devt	dv	Indicate lan? guage/codeset sup? port
display_pc_char	dispc	S1	Display PC charac? ter #1
end_bit_image_region	endbi	Yy	End a bit-image region
enter_pc_charset_mode	smpch	S2	Enter PC character display mode
enter_scancode_mode	smsc	S4	Enter PC scancode mode
exit_pc_charset_mode	rmpch	S3	Exit PC character display mode
exit_scancode_mode	rmsc	S5	Exit PC scancode

			mode
get_mouse	getm	Gm	Curses should get button events, pa? rameter #1 not documented.
key_mouse	kmous	Km	Mouse event has occurred
mouse_info	minfo	Mi	Mouse status in? formation
pc_term_options	pctrm	S6	PC terminal op? tions
pkey_plab	pfxl	xl	Program function key #1 to type string #2 and show string #3
req_mouse_pos	reqmp	RQ	Request mouse po? sition
scancode_escape	scesc	S7	Escape for scan? code emulation
set0_des_seq	s0ds	s0	Shift to codeset 0 (EUC set 0, ASCII)
set1_des_seq	s1ds	s1	Shift to codeset 1
set2_des_seq	s2ds	s2	Shift to codeset 2
set3_des_seq	s3ds	s3	Shift to codeset 3
set_a_background	setab	AB	Set background color to #1, using ANSI escape
set_a_foreground	setaf	AF	Set foreground color to #1, using ANSI escape
set_color_band	setcolor	Yz	Change to ribbon color #1
set_lr_margin	smglr	ML	Set both left and

			right margins to #1, #2. (ML is not in BSD term? cap).
set_page_length	slines	YZ	Set page length to #1 lines
set_tb_margin	smgtb	MT	Sets both top and bottom margins to #1, #2

The XSI Curses standard added these hardcopy capabilities. They were used in some post-4.1 versions of System V curses, e.g., Solaris 2.5 and IRIX 6.x. Except for YI, the ncurses termcap names for them are invented. According to the XSI Curses standard, they have no termcap names. If your compiled terminfo entries use these, they may not be binary-compatible with System V terminfo entries after SVr4.1; beware!

Variable String	Cap-name	TCap Code	Description
enter_horizontal_hl_mode	ehhlm	Xh	Enter horizontal highlight mode
enter_left_hl_mode	elhlm	Xl	Enter left highlight mode
enter_low_hl_mode	elohlm	Xo	Enter low highlight mode
enter_right_hl_mode	erhlm	Xr	Enter right high? light mode
enter_top_hl_mode	ethlm	Xt	Enter top highlight mode
enter_vertical_hl_mode	evhlm	Xv	Enter vertical high? light mode
set_a_attributes	sgr1	sA	Define second set of video attributes #1-#6
set_pglen_inch	slength	YI	Set page length to

#1 hundredth of an
inch (some implemen?
tations use sL for
termcap).

User-Defined Capabilities

The preceding section listed the predefined capabilities. They deal with some special features for terminals no longer (or possibly never) produced. Occasionally there are special features of newer terminals which are awkward or impossible to represent by reusing the predefined capabilities.

ncurses addresses this limitation by allowing user-defined capabili? ties. The tic and infocmp programs provide the -x option for this pur? pose. When -x is set, tic treats unknown capabilities as user-defined. That is, if tic encounters a capability name which it does not recog? nize, it infers its type (boolean, number or string) from the syntax and makes an extended table entry for that capability. The use_ex? tended_names(3X) function makes this information conditionally avail? able to applications. The ncurses library provides the data leaving most of the behavior to applications:

? User-defined capability strings whose name begins with ?k? are treated as function keys.

? The types (boolean, number, string) determined by tic can be in? ferred by successful calls on tigetflag, etc.

? If the capability name happens to be two characters, the capability is also available through the termcap interface.

While termcap is said to be extensible because it does not use a prede? fined set of capabilities, in practice it has been limited to the capa? bilities defined by terminfo implementations. As a rule, user-defined capabilities intended for use by termcap applications should be limited to booleans and numbers to avoid running past the 1023 byte limit as? sumed by termcap implementations and their applications. In particu? lar, providing extended sets of function keys (past the 60 numbered keys and the handful of special named keys) is best done using the

longer names available using terminfo.

A Sample Entry

The following entry, describing an ANSI-standard terminal, is represen?
tative of what a terminfo entry for a modern terminal typically looks
like.

```
ansi|ansi/pc-term compatible with color,
        am, mc5i, mir, msgr,
        colors#8, cols#80, it#8, lines#24, ncv#3, pairs#64,
        acsc=+\020\,\021-\030.^Y0\333`\004a\261f\370g\361h\260
            j\331k\277l\332m\300n\305o~p\304q\304r\304s_t\303
            u\264v\301w\302x\263y\363z\362{\343|\330}\234~\376,
        bel=^G, blink=\E[5m, bold=\E[1m, cbt=\E[Z, clear=\E[H\E[J,
        cr=^M, cub=\E[%p1%dD, cub1=\E[D, cud=\E[%p1%dB, cud1=\E[B,
        cuf=\E[%p1%dC, cuf1=\E[C, cup=\E[%i%p1%d;%p2%dH,
        cuu=\E[%p1%dA, cuu1=\E[A, dch=\E[%p1%dP, dch1=\E[P,
        dl=\E[%p1%dM, dl1=\E[M, ech=\E[%p1%dX, ed=\E[J, el=\E[K,
        el1=\E[1K, home=\E[H, hpa=\E[%i%p1%dG, ht=\E[I, hts=\EH,
        ich=\E[%p1%d@, il=\E[%p1%dL, il1=\E[L, ind=^J,
        indn=\E[%p1%dS, invis=\E[8m, kbs=^H, kcbt=\E[Z, kcub1=\E[D,
        kcud1=\E[B, kcuf1=\E[C, kcuu1=\E[A, khome=\E[H, kich1=\E[L,
        mc4=\E[4i, mc5=\E[5i, nel=\r\E[S, op=\E[39;49m,
        rep=%p1%c\E[%p2%{1}%-%db, rev=\E[7m, rin=\E[%p1%dT,
        rmacs=\E[10m, rmpch=\E[10m, rmso=\E[m, rmul=\E[m,
        s0ds=\E(B, s1ds=\E)B, s2ds=\E*B, s3ds=\E+B,
        setab=\E[4%p1%dm, setaf=\E[3%p1%dm,
        sgr=\E[0;10%?%p1%t;7%;
                %?%p2%t;4%;
                %?%p3%t;7%;
                %?%p4%t;5%;
                %?%p6%t;1%;
                %?%p7%t;8%;
                %?%p9%t;11%;m,
        sgr0=\E[0;10m, smacs=\E[11m, smpch=\E[11m, smso=\E[7m,
```

smul=\E[4m, tbc=\E[3g, u6=\E[%i%d;%dR, u7=\E[6n,

u8=\E[?%[;0123456789]c, u9=\E[c, vpa=\E[%i%p1%dd,

Entries may continue onto multiple lines by placing white space at the beginning of each line except the first. Comments may be included on lines beginning with ?#?. Capabilities in terminfo are of three types:

? Boolean capabilities which indicate that the terminal has some par? ticular feature,

? numeric capabilities giving the size of the terminal or the size of particular delays, and

? string capabilities, which give a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have names. For instance, the fact that ANSI-standard terminals have automatic margins (i.e., an automatic return and line-feed when the end of a line is reached) is indicated by the capability am. Hence the description of ansi includes am. Numeric capabilities are followed by the character ?#? and then a positive value. Thus cols, which indicates the number of columns the terminal has, gives the value ?80? for ansi. Values for numeric capabilities may be specified in decimal, octal or hexadecimal, using the C programming language con? ventions (e.g., 255, 0377 and 0xff or 0xFF).

Finally, string valued capabilities, such as el (clear to end of line sequence) are given by the two-character code, an ?=?, and then a string ending at the next following ?,?.

A number of escape sequences are provided in the string valued capabil? ities for easy encoding of characters there:

? Both \E and \e map to an ESCAPE character,

? ^x maps to a control-x for any appropriate x, and

? the sequences

\n, \l, \r, \t, \b, \f, and \s

produce

newline, line-feed, return, tab, backspace, form-feed, and space, respectively.

X/Open Curses does not say what ?appropriate x? might be. In practice, that is a printable ASCII graphic character. The special case ?^?? is interpreted as DEL (127). In all other cases, the character value is AND'd with 0x1f, mapping to ASCII control codes in the range 0 through 31.

Other escapes include

? \^ for ^,

? \\ for \,

? \, for comma,

? \: for :,

? and \0 for null.

\0 will produce \200, which does not terminate a string but behaves as a null character on most terminals, providing CS7 is specified. See stty(1).

The reason for this quirk is to maintain binary compatibility of the compiled terminfo files with other implementations, e.g., the SVr4 systems, which document this. Compiled terminfo files use null-terminated strings, with no lengths. Modifying this would re? quire a new binary format, which would not work with other imple? mentations.

Finally, characters may be given as three octal digits after a \.

A delay in milliseconds may appear anywhere in a string capability, en? closed in $<..> brackets, as in el=\EK$<5>, and padding characters are supplied by tputs(3X) to provide this delay.

? The delay must be a number with at most one decimal place of preci? sion; it may be followed by suffixes ?*? or ?/? or both.

? A ?*? indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert character, the factor is still the number of lines affected.)

Normally, padding is advisory if the device has the xon capability; it is used for cost computation but does not trigger delays.

? A ?/? suffix indicates that the padding is mandatory and forces a

delay of the given number of milliseconds even on devices for which xon is present to indicate flow control.

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the second ind in the example above.

Fetching Compiled Descriptions

The ncurses library searches for terminal descriptions in several places. It uses only the first description found. The library has a compiled-in list of places to search which can be overridden by envi? ronment variables. Before starting to search, ncurses eliminates du? plicates in its search list.

? If the environment variable TERMINFO is set, it is interpreted as the pathname of a directory containing the compiled description you are working on. Only that directory is searched.

? If TERMINFO is not set, ncurses will instead look in the directory $HOME/.terminfo for a compiled description.

? Next, if the environment variable TERMINFO_DIRS is set, ncurses will interpret the contents of that variable as a list of colon-separated directories (or database files) to be searched.

An empty directory name (i.e., if the variable begins or ends with a colon, or contains adjacent colons) is interpreted as the system location /usr/share/terminfo.

? Finally, ncurses searches these compiled-in locations:

? a list of directories (no default value), and

? the system terminfo directory, /usr/share/terminfo (the com? piled-in default).

Preparing Descriptions

We now outline how to prepare descriptions of terminals. The most ef? fective way to prepare a terminal description is by imitating the de? scription of a similar terminal in terminfo and to build up a descrip? tion gradually, using partial descriptions with vi or some other screen-oriented program to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the

terminfo file to describe it or bugs in the screen-handling code of the test program.

To get the padding for insert line right (if the terminal manufacturer did not document it) a severe test is to edit a large file at 9600 baud, delete 16 or so lines from the middle of the screen, then hit the ?u? key several times quickly. If the terminal messes up, more padding is usually needed. A similar test can be used for insert character.

Basic Capabilities

The number of columns on each line for the terminal is given by the cols numeric capability. If the terminal is a CRT, then the number of lines on the screen is given by the lines capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the am capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the clear string capability. If the terminal over? strikes (rather than clearing a position when a character is struck over) then it should have the os capability. If the terminal is a printing terminal, with no soft copy unit, give it both hc and os. (os applies to storage scope terminals, such as TEKTRONIX 4010 series, as well as hard copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as cr. (Normally this will be carriage return, control/M.) If there is a code to pro? duce an audible signal (bell, beep, etc) give this as bel.

If there is a code to move the cursor one position to the left (such as backspace) that capability should be given as cub1. Similarly, codes to move to the right, up, and down should be given as cuf1, cuu1, and cud1. These local cursor motions should not alter the text they pass over, for example, you would not normally use ?cuf1= ? because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in terminfo are undefined at the left and top edges of a CRT terminal. Programs should never attempt to backspace around the left edge, unless bw is given, and never attempt to go up locally off the top. In order

to scroll text up, a program will go to the bottom left corner of the screen and send the ind (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the ri (reverse index) string. The strings ind and ri are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are indn and rin which have the same semantics as ind and ri except that they take one parameter, and scroll that many lines. They are also undefined except at the appropriate edge of the screen.

The am capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a cuf1 from the last column. The only local motion which is defined from the left edge is if bw is given, then a cub1 from the left edge will move to the right edge of the previous row. If bw is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch selectable automatic margins, the terminfo file usually assumes that this is on; i.e., am. If the terminal has a command which moves to the first col? umn of the next line, that command can be given as nel (newline). It does not matter if the command clears the remainder of the current line, so if the terminal has no cr and lf it may still be possible to craft a working nel out of one or both of them.

These capabilities suffice to describe hard-copy and ?glass-tty? termi? nals. Thus the model 33 teletype is described as

33|tty33|tty|model 33 teletype,

bel=^G, cols#72, cr=^M, cud1=^J, hc, ind=^J, os,

while the Lear Siegler ADM-3 is described as

adm3|3|lsi adm3,

am, bel=^G, clear=^Z, cols#80, cr=^M, cub1=^H, cud1=^J,

ind=^J, lines#24,

Parameterized Strings

Cursor addressing and other strings requiring parameters in the termi? nal are described by a parameterized string capability, with printf-

like escapes such as %x in it. For example, to address the cursor, the cup capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory relative cursor addressing, that can be indicated by mrcup.

The parameter mechanism uses a stack and special % codes to manipulate it. Typically a sequence will push one of the parameters onto the stack and then print it in some format. Print (e.g., "%d") is a spe? cial case. Other operations, including "%t" pop their operand from the stack. It is noted that more complex operations are often necessary, e.g., in the sgr string.

The % encodings have the following meanings:

%% outputs ?%?

%[[:]flags][width[.precision]][doxXs]

as in printf(3), flags are [-+#] and space. Use a ?:? to allow the next character to be a ?-? flag, avoiding interpreting ?%-? as an operator.

%c print pop() like %c in printf

%s print pop() like %s in printf

%p[1-9]

push i'th parameter

%P[a-z]

set dynamic variable [a-z] to pop()

%g[a-z]/

get dynamic variable [a-z] and push it

%P[A-Z]

set static variable [a-z] to pop()

%g[A-Z]

get static variable [a-z] and push it

The terms ?static? and ?dynamic? are misleading. Historically, these are simply two different sets of variables, whose values are not reset between calls to tparm(3X). However, that fact is not

documented in other implementations. Relying on it will adversely impact portability to other implementations.

%'c' char constant c

%{nn}

integer constant nn

%l push strlen(pop)

%+, %-, %*, %/, %m

arithmetic (%m is mod): push(pop() op pop())

%&, %|, %^

bit operations (AND, OR and exclusive-OR): push(pop() op pop())

%=, %>, %<

logical operations: push(pop() op pop())

%A, %O

logical AND and OR operations (for conditionals)

%!, %~

unary operations (logical and bit complement): push(op pop())

%i add 1 to first two parameters (for ANSI terminals)

%? expr %t thenpart %e elsepart %;

This forms an if-then-else. The %e elsepart is optional. Usually the %? expr part pushes a value onto the stack, and %t pops it from the stack, testing if it is nonzero (true). If it is zero (false), control passes to the %e (else) part.

It is possible to form else-if's a la Algol 68:

%? c1 %t b1 %e c2 %t b2 %e c3 %t b3 %e c4 %t b4 %e %;

where ci are conditions, bi are bodies.

Use the -f option of tic or infocmp to see the structure of if-then-else's. Some strings, e.g., sgr can be very complicated when written on one line. The -f option splits the string into lines with the parts indented.

Binary operations are in postfix form with the operands in the usual order. That is, to get x-5 one would use "%gx%{5}%-". %P and %g vari? ables are persistent across escape-string evaluations.

Consider the HP2645, which, to get to row 3 and column 12, needs to be

sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its cup capability is ?cup=6\E&%p2%2dc%p1%2dY?.

The Microterm ACT-IV needs the current row and column sent preceded by a ^T, with the row and column simply encoded in binary, ?cup=^T%p1%c%p2%c?. Terminals which use ?%c? need to be able to backspace the cursor (cub1), and to move the cursor up one line on the screen (cuu1). This is necessary because it is not always safe to transmit \n ^D and \r, as the system may change or discard them. (The library routines dealing with terminfo set tty modes so that tabs are never expanded, so \t is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus ?cup=\E=%p1%' '%+%c%p2%' '%+%c?. After sending ?\E=?, this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values) and outputs that value as a character. Then the same is done for the second parameter. More complex arithmetic is possible using the stack.

Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen) then this can be given as home; similarly a fast way of getting to the lower left-hand corner can be given as ll; this may involve going up with cuu1 from the home position, but a program should never do this itself (unless ll does) because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as addressing to (0,0): to the top left cor? ner of the screen, not of memory. (Thus, the \EH sequence on HP termi? nals cannot be used for home.)

If the terminal has row or column absolute cursor addressing, these can be given as single parameter capabilities hpa (horizontal position ab? solute) and vpa (vertical position absolute). Sometimes these are

shorter than the more general two parameter sequence (as with the hp2645) and can be used in preference to cup. If there are parameter? ized local motions (e.g., move n spaces to the right) these can be given as cud, cub, cuf, and cuu with a single parameter indicating how many spaces to move. These are primarily useful if the terminal does not have cup, such as the TEKTRONIX 4025.

If the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as smcup and rmcup. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cur? sor addressing, a one screen-sized window must be fixed into the termi? nal for cursor addressing to work properly. This is also used for the TEKTRONIX 4025, where smcup sets the command character to be the one used by terminfo. If the smcup sequence will not restore the screen after an rmcup sequence is output (to the state prior to outputting rm? cup), specify nrrmc.

Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as el. If the terminal can clear from the beginning of the line to the current position inclusive, leaving the cursor where it is, this should be given as el1. If the terminal can clear from the current position to the end of the display, then this should be given as ed. Ed is only defined from the first column of a line. (Thus, it can be simulated by a request to delete a large number of lines, if a true ed is not avail? able.)

Insert/delete line and vertical motions

If the terminal can open a new blank line before the line where the cursor is, this should be given as il1; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as dl1; this is done only from the first

position on the line to be deleted. Versions of il1 and dl1 which take a single parameter and insert or delete that many lines can be given as il and dl.

If the terminal has a settable scrolling region (like the vt100) the command to set this can be described with the csr capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command.

It is possible to get the effect of insert or delete line using csr on a properly chosen region; the sc and rc (save and restore cursor) com? mands may be useful for ensuring that your synthesized insert/delete string does not move the cursor. (Note that the ncurses(3X) library does this synthesis automatically, so you need not compose in? sert/delete strings for an entry with csr).

Yet another way to construct insert and delete might be to use a combi? nation of index with the memory-lock feature found on some terminals (like the HP-700/90 series, which however also has insert/delete).

Inserting lines at the top or bottom of the screen can also be done us? ing ri or ind on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

The boolean non_dest_scroll_region should be set if each scrolling win? dow is effectively a view port on a screen-sized canvas. To test for this capability, create a scrolling region in the middle of the screen, write something to the bottom line, move the cursor to the top of the region, and do ri followed by dl1 or ind. If the data scrolled off the bottom of the region by the ri re-appears, then scrolling is non-de? structive. System V and XSI Curses expect that ind, ri, indn, and rin will simulate destructive scrolling; their documentation cautions you not to define csr unless this is true. This curses implementation is more liberal and will do explicit erases after scrolling if ndsrc is defined.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the parameterized string wind. The four parameters are the starting and ending lines in

memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, then the da capability should be given; if display memory can be retained below, then db should be given. These indicate that deleting a line or scrolling may bring non-blank lines up from below or that scrolling back with ri may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to in? sert/delete character which can be described using terminfo. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly.

Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks.

You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type ?abc def? using local cursor motions (not spaces) between the ?abc? and the ?def?. Then position the cursor before the ?abc? and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the ?abc? shifts over to the ?def? which then move together around the end of the current line and onto the next as you insert, you have the sec? ond type of terminal, and should give the capability in, which stands for ?insert null?.

While these are two logically separate attributes (one line versus multi-line insert mode, and special treatment of untyped spaces) we have seen no terminals whose insert mode cannot be described with the single attribute.

Terminfo can describe both terminals which have an insert mode, and terminals which send a simple sequence to open a blank position on the current line. Give as smir the sequence to get into insert mode. Give

as rmir the sequence to leave insert mode. Now give as ich1 any se? quence needed to be sent just before sending the character to be in? serted. Most terminals with a true insert mode will not give ich1; terminals which send a sequence to open a screen position should give it here.

If your terminal has both, insert mode is usually preferable to ich1. Technically, you should not give both unless the terminal actually re? quires both to be used in combination. Accordingly, some non-curses applications get confused if both are present; the symptom is doubled characters in an update using insert. This requirement is now rare; most ich sequences do not require previous smir, and most smir insert modes do not require ich1 before each character. Therefore, the new curses actually assumes this is the case and uses either rmir/smir or ich/ich1 as appropriate (but not both). If you have to write an entry to be used under new curses for a terminal old enough to need both, in? clude the rmir/smir sequences in ich1.

If post insert padding is needed, give this as a number of milliseconds in ip (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in ip. If your terminal needs both to be placed into an ?insert mode? and a special code to precede each inserted character, then both smir/rmir and ich1 can be given, and both will be used. The ich capability, with one pa? rameter, n, will repeat the effects of ich1 n times.

If padding is necessary between characters typed while not in insert mode, give this as a number of milliseconds padding in rmp.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g., if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability mir to speed up inserting in this case. Omitting mir will affect only speed. Some terminals (notably Datamedia's) must not have mir because of the way their insert mode works.

Finally, you can specify dch1 to delete a single character, dch with

one parameter, n, to delete n characters, and delete mode by giving smdc and rmdc to enter and exit delete mode (any mode the terminal needs to be placed in for dch1 to work).

A command to erase n characters (equivalent to outputting n blanks without moving the cursor) can be given as ech with one parameter.

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as standout mode, representing a good, high contrast, easy-on-the-eyes, format for highlighting error messages and other at? tention getters. (If you have a choice, reverse video plus half-bright is good, or reverse video alone.) The sequences to enter and exit standout mode are given as smso and rmso, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then xmc should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as smul and rmul respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Mi? croterm Mime, this can be given as uc.

Other capabilities to enter various highlighting modes include blink (blinking) bold (bold or extra bright) dim (dim or half-bright) invis (blanking or invisible text) prot (protected) rev (reverse video) sgr0 (turn off all attribute modes) smacs (enter alternate character set mode) and rmacs (exit alternate character set mode). Turning on any of these modes singly may or may not turn off other modes.

If there is a sequence to set arbitrary combinations of modes, this should be given as sgr (set attributes), taking 9 parameters. Each pa? rameter is either 0 or nonzero, as the corresponding attribute is on or off. The 9 parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, alternate character set. Not all modes need be supported by sgr, only those for which corresponding sep? arate attribute commands exist.

For example, the DEC vt220 supports most of the modes:

tparm parameter	attribute	escape sequence
none	none	\E[0m
p1	standout	\E[0;1;7m
p2	underline	\E[0;4m
p3	reverse	\E[0;7m
p4	blink	\E[0;5m
p5	dim	not available
p6	bold	\E[0;1m
p7	invis	\E[0;8m
p8	protect	not used
p9	altcharset	^O (off) ^N (on)

We begin each escape sequence by turning off any existing modes, since there is no quick way to determine whether they are active. Standout is set up to be the combination of reverse and bold. The vt220 termi? nal has a protect mode, though it is not commonly used in sgr because it protects characters on the screen from the host's erasures. The altcharset mode also is different in that it is either ^O or ^N, de? pending on whether it is off or on. If all modes are turned on, the resulting sequence is \E[0;1;4;5;7;8m^N.

Some sequences are common to different modes. For example, ;7 is out? put when either p1 or p3 is true, that is, if either standout or re? verse modes are turned on.

Writing out the above sequences, along with their dependencies yields

sequence	when to output	terminfo translation
\E[0	always	\E[0
;1	if p1 or p6	%?%p1%p6%|%t;1%;
;4	if p2	%?%p2%|%t;4%;
;5	if p4	%?%p4%|%t;5%;
;7	if p1 or p3	%?%p1%p3%|%t;7%;
;8	if p7	%?%p7%|%t;8%;
m	always	m
^N or ^O	if p9 ^N, else ^O	%?%p9%t^N%e^O%;

Putting this all together into the sgr sequence gives:

sgr=\E[0%?%p1%p6%|%t;1%;%?%p2%t;4%;%?%p4%t;5%;

%?%p1%p3%|%t;7%;%?%p7%t;8%;m%?%p9%t\016%e\017%;,

Remember that if you specify sgr, you must also specify sgr0. Also, some implementations rely on sgr being given if sgr0 is, Not all ter? minfo entries necessarily have an sgr string, however. Many terminfo entries are derived from termcap entries which have no sgr string. The only drawback to adding an sgr string is that termcap also assumes that sgr0 does not exit alternate character set mode.

Terminals with the ?magic cookie? glitch (xmc) deposit special ?cook? ies? when they receive mode-setting sequences, which affect the display algorithm rather than having extra bits for each character. Some ter? minals, such as the HP 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline, unless the msgr capability, asserting that it is safe to move in standout mode, is present.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given as flash; it must not move the cursor.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to make, for example, a non-blinking underline into an easier to find block or blinking underline) give this sequence as cvvis. If there is a way to make the cursor completely invisible, give that as civis. The capability cnorm should be given which undoes the effects of both of these modes.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability ul. If a character overstriking another leaves both characters on the screen, specify the capability os. If overstrikes are erasable with a blank, then this should be indicated by giving eo.

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as smkx and rmkx. Other? wise the keypad is assumed to always transmit.

The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as kcub1, kcuf1, kcuu1, kcud1, and khome re? spectively. If there are function keys such as f0, f1, ..., f10, the codes they send can be given as kf0, kf1, ..., kf10. If these keys have labels other than the default f0 through f10, the labels can be given as lf0, lf1, ..., lf10.

The codes transmitted by certain other special keys can be given:

? kll (home down),

? kbs (backspace),

? ktbc (clear all tabs),

? kctab (clear the tab stop in this column),

? kclr (clear screen or erase key),

? kdch1 (delete character),

? kdl1 (delete line),

? krmir (exit insert mode),

? kel (clear to end of line),

? ked (clear to end of screen),

? kich1 (insert character or enter insert mode),

? kil1 (insert line),

? knp (next page),

? kpp (previous page),

? kind (scroll forward/down),

? kri (scroll backward/up),

? khts (set a tab stop in this column).

In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, the other five keys can be given as ka1, ka3, kb2, kc1, and kc3. These keys are useful when the effects of a 3 by 3 di?

rectional pad are needed.

Strings to program function keys can be given as pfkey, pfloc, and pfx. A string to program screen labels should be specified as pln. Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal dependent manner. The difference between the capabilities is that pfkey causes pressing the given key to be the same as the user typing the given string; pfloc causes the string to be executed by the terminal in lo? cal; and pfx causes the string to be transmitted to the computer.

The capabilities nlab, lw and lh define the number of programmable screen labels and their width and height. If there are commands to turn the labels on and off, give them in smln and rmln. smln is nor? mally output after one or more pln sequences to make sure that the change becomes visible.

Tabs and Initialization

A few capabilities are used only for tabs:

? If the terminal has hardware tabs, the command to advance to the next tab stop can be given as ht (usually control/I).

? A ?back-tab? command which moves leftward to the preceding tab stop can be given as cbt.

By convention, if the teletype modes indicate that tabs are being expanded by the computer rather than being sent to the terminal, programs should not use ht or cbt even if they are present, since the user may not have the tab stops properly set.

? If the terminal has hardware tabs which are initially set every n spaces when the terminal is powered up, the numeric parameter it is given, showing the number of spaces the tabs are set to.

The it capability is normally used by the tset command to determine whether to set the mode for hardware tab expansion, and whether to set the tab stops. If the terminal has tab stops that can be saved in non-volatile memory, the terminfo description can assume that they are properly set.

Other capabilities include

? is1, is2, and is3, initialization strings for the terminal,

? iprog, the path name of a program to be run to initialize the ter?
minal,

? and if, the name of a file containing long initialization strings.

These strings are expected to set the terminal into modes consistent with the rest of the terminfo description. They are normally sent to the terminal, by the init option of the tput program, each time the user logs in. They will be printed in the following order:

run the program

iprog

output

is1 and

is2

set the margins using

mgc or

smglp and smgrp or

smgl and smgr

set tabs using

tbc and hts

print the file

if

and finally output

is3.

Most initialization is done with is2. Special terminal modes can be set up without duplicating strings by putting the common sequences in is2 and special cases in is1 and is3.

A set of sequences that does a harder reset from a totally unknown state can be given as rs1, rs2, rf and rs3, analogous to is1 , is2 , if and is3 respectively. These strings are output by reset option of tput, or by the reset program (an alias of tset), which is used when the terminal gets into a wedged state. Commands are normally placed in rs1, rs2 rs3 and rf only if they produce annoying effects on the screen

and are not necessary when logging in. For example, the command to set the vt100 into 80-column mode would normally be part of is2, but it causes an annoying glitch of the screen and is not normally needed since the terminal is usually already in 80-column mode.

The reset program writes strings including iprog, etc., in the same or? der as the init program, using rs1, etc., instead of is1, etc. If any of rs1, rs2, rs3, or rf reset capability strings are missing, the reset program falls back upon the corresponding initialization capability string.

If there are commands to set and clear tab stops, they can be given as tbc (clear all tab stops) and hts (set a tab stop in the current column of every row). If a more complex sequence is needed to set the tabs than can be described by this, the sequence can be placed in is2 or if.

The tput reset command uses the same capability strings as the reset command, although the two programs (tput and reset) provide different command-line options.

In practice, these terminfo capabilities are not often used in initial? ization of tabs (though they are required for the tabs program):

? Almost all hardware terminals (at least those which supported tabs) initialized those to every eight columns:

The only exception was the AT&T 2300 series, which set tabs to ev? ery five columns.

? In particular, developers of the hardware terminals which are com? monly used as models for modern terminal emulators provided docu? mentation demonstrating that eight columns were the standard.

? Because of this, the terminal initialization programs tput and tset use the tbc (clear_all_tabs) and hts (set_tab) capabilities di? rectly only when the it (init_tabs) capability is set to a value other than eight.

Delays and Padding

Many older and slower terminals do not support either XON/XOFF or DTR handshaking, including hard copy terminals and some very archaic CRTs (including, for example, DEC VT100s). These may require padding char?

acters after certain cursor motions and screen changes.

If the terminal uses xon/xoff handshaking for flow control (that is, it automatically emits ^S back to the host when its input buffers are close to full), set xon. This capability suppresses the emission of padding. You can also set it for memory-mapped console devices effec? tively that do not have a speed limit. Padding information should still be included so that routines can make better decisions about rel? ative costs, but actual pad characters will not be transmitted.

If pb (padding baud rate) is given, padding is suppressed at baud rates below the value of pb. If the entry has no padding baud rate, then whether padding is emitted or not is completely controlled by xon.

If the terminal requires other than a null (zero) character as a pad, then this can be given as pad. Only the first character of the pad string is used.

Status Lines

Some terminals have an extra ?status line? which is not normally used by software (and thus not counted in the terminal's lines capability).

The simplest case is a status line which is cursor-addressable but not part of the main scrolling region on the screen; the Heathkit H19 has a status line of this kind, as would a 24-line VT100 with a 23-line scrolling region set up on initialization. This situation is indicated by the hs capability.

Some terminals with status lines need special sequences to access the status line. These may be expressed as a string with single parameter tsl which takes the cursor to a given zero-origin column on the status line. The capability fsl must return to the main-screen cursor posi? tions before the last tsl. You may need to embed the string values of sc (save cursor) and rc (restore cursor) in tsl and fsl to accomplish this.

The status line is normally assumed to be the same width as the width of the terminal. If this is untrue, you can specify it with the nu? meric capability wsl.

A command to erase or blank the status line may be specified as dsl.

The boolean capability eslok specifies that escape sequences, tabs, etc., work ordinarily in the status line.

The ncurses implementation does not yet use any of these capabilities. They are documented here in case they ever become important.

Line Graphics

Many terminals have alternate character sets useful for forms-drawing. Terminfo and curses have built-in support for most of the drawing char? acters supported by the VT100, with some characters from the AT&T 4410v1 added. This alternate character set may be specified by the acsc capability.

Glyph Name	ACS Name	Ascii Default	acsc Char	acsc Value
arrow pointing right	ACS_RARROW	>	+	0x2b
arrow pointing left	ACS_LARROW	<	,	0x2c
arrow pointing up	ACS_UARROW	^	-	0x2d
arrow pointing down	ACS_DARROW	v	.	0x2e
solid square block	ACS_BLOCK	#	0	0x30
diamond	ACS_DIAMOND	+	`	0x60
checker board (stipple)	ACS_CKBOARD	:	a	0x61
degree symbol	ACS_DEGREE	\	f	0x66
plus/minus	ACS_PLMINUS	#	g	0x67
board of squares	ACS_BOARD	#	h	0x68
lantern symbol	ACS_LANTERN	#	i	0x69
lower right corner	ACS_LRCORNER	+	j	0x6a
upper right corner	ACS_URCORNER	+	k	0x6b
upper left corner	ACS_ULCORNER	+	l	0x6c
lower left corner	ACS_LLCORNER	+	m	0x6d
large plus or crossover	ACS_PLUS	+	n	0x6e
scan line 1	ACS_S1	~	o	0x6f
scan line 3	ACS_S3	-	p	0x70
horizontal line	ACS_HLINE	-	q	0x71
scan line 7	ACS_S7	-	r	0x72

scan line 9	ACS_S9	_	s	0x73
tee pointing right	ACS_LTEE	+	t	0x74
tee pointing left	ACS_RTEE	+	u	0x75
tee pointing up	ACS_BTEE	+	v	0x76
tee pointing down	ACS_TTEE	+	w	0x77
vertical line	ACS_VLINE	|	x	0x78
less-than-or-equal-to	ACS_LEQUAL	<	y	0x79
greater-than-or-equal-to	ACS_GEQUAL	>	z	0x7a
greek pi	ACS_PI	*	{	0x7b
not-equal	ACS_NEQUAL	!	|	0x7c
UK pound sign	ACS_STERLING	f	}	0x7d
bullet	ACS_BULLET	o	~	0x7e

A few notes apply to the table itself:

? X/Open Curses incorrectly states that the mapping for lantern is uppercase ?I? although Unix implementations use the lowercase ?i? mapping.

? The DEC VT100 implemented graphics using the alternate character set feature, temporarily switching modes and sending characters in the range 0x60 (96) to 0x7e (126) (the acsc Value column in the ta? ble).

? The AT&T terminal added graphics characters outside that range. Some of the characters within the range do not match the VT100; presumably they were used in the AT&T terminal: board of squares replaces the VT100 newline symbol, while lantern symbol replaces the VT100 vertical tab symbol. The other VT100 symbols for control characters (horizontal tab, carriage return and line-feed) are not (re)used in curses.

The best way to define a new device's graphics set is to add a column to a copy of this table for your terminal, giving the character which (when emitted between smacs/rmacs switches) will be rendered as the corresponding graphic. Then read off the VT100/your terminal character pairs right to left in sequence; these become the ACSC string.

The curses library functions init_pair and init_color manipulate the color pairs and color values discussed in this section (see curs_color(3X) for details on these and related functions).

Most color terminals are either ?Tektronix-like? or ?HP-like?:

? Tektronix-like terminals have a predefined set of N colors (where N is usually 8), and can set character-cell foreground and background characters independently, mixing them into N * N color-pairs.

? On HP-like terminals, the user must set each color pair up sepa? rately (foreground and background are not independently settable). Up to M color-pairs may be set up from 2*M different colors. ANSI-compatible terminals are Tektronix-like.

Some basic color capabilities are independent of the color method. The numeric capabilities colors and pairs specify the maximum numbers of colors and color-pairs that can be displayed simultaneously. The op (original pair) string resets foreground and background colors to their default values for the terminal. The oc string resets all colors or color-pairs to their default values for the terminal. Some terminals (including many PC terminal emulators) erase screen areas with the cur? rent background color rather than the power-up default background; these should have the boolean capability bce.

While the curses library works with color pairs (reflecting the inabil? ity of some devices to set foreground and background colors indepen? dently), there are separate capabilities for setting these features:

? To change the current foreground or background color on a Tek? tronix-type terminal, use setaf (set ANSI foreground) and setab (set ANSI background) or setf (set foreground) and setb (set back? ground). These take one parameter, the color number. The SVr4 documentation describes only setaf/setab; the XPG4 draft says that "If the terminal supports ANSI escape sequences to set background and foreground, they should be coded as setaf and setab, respec? tively.

? If the terminal supports other escape sequences to set background and foreground, they should be coded as setf and setb, respec?

tively. The vidputs and the refresh(3X) functions use the setaf and setab capabilities if they are defined.

The setaf/setab and setf/setb capabilities take a single numeric argument each. Argument values 0-7 of setaf/setab are portably defined as follows (the middle column is the symbolic #define available in the header for the curses or ncurses libraries). The terminal hardware is free to map these as it likes, but the RGB values indicate normal locations in color space.

Color	#define	Value	RGB
black	COLOR_BLACK	0	0, 0, 0
red	COLOR_RED	1	max,0,0
green	COLOR_GREEN	2	0,max,0
yellow	COLOR_YELLOW	3	max,max,0
blue	COLOR_BLUE	4	0,0,max
magenta	COLOR_MAGENTA	5	max,0,max
cyan	COLOR_CYAN	6	0,max,max
white	COLOR_WHITE	7	max,max,max

The argument values of setf/setb historically correspond to a different mapping, i.e.,

Color	#define	Value	RGB
black	COLOR_BLACK	0	0, 0, 0
blue	COLOR_BLUE	1	0,0,max
green	COLOR_GREEN	2	0,max,0
cyan	COLOR_CYAN	3	0,max,max
red	COLOR_RED	4	max,0,0
magenta	COLOR_MAGENTA	5	max,0,max
yellow	COLOR_YELLOW	6	max,max,0
white	COLOR_WHITE	7	max,max,max

It is important to not confuse the two sets of color capabilities; otherwise red/blue will be interchanged on the display.

On an HP-like terminal, use scp with a color-pair number parameter to set which color pair is current.

Some terminals allow the color values to be modified:

? On a Tektronix-like terminal, the capability ccc may be present to indicate that colors can be modified. If so, the initc capability will take a color number (0 to colors - 1)and three more parameters which describe the color. These three parameters default to being interpreted as RGB (Red, Green, Blue) values. If the boolean capability hls is present, they are instead as HLS (Hue, Lightness, Saturation) indices. The ranges are terminal-dependent.

? On an HP-like terminal, initp may give a capability for changing a color-pair value. It will take seven parameters; a color-pair number (0 to max_pairs - 1), and two triples describing first background and then foreground colors. These parameters must be (Red, Green, Blue) or (Hue, Lightness, Saturation) depending on hls.

On some color terminals, colors collide with highlights. You can register these collisions with the ncv capability. This is a bit-mask of attributes not to be used when colors are enabled. The correspondence with the attributes understood by curses is as follows:

Attribute	Bit	Decimal	Set by
A_STANDOUT	0	1	sgr
A_UNDERLINE	1	2	sgr
A_REVERSE	2	4	sgr
A_BLINK	3	8	sgr
A_DIM	4	16	sgr
A_BOLD	5	32	sgr
A_INVIS	6	64	sgr
A_PROTECT	7	128	sgr
A_ALTCHARSET	8	256	sgr
A_HORIZONTAL	9	512	sgr1
A_LEFT	10	1024	sgr1
A_LOW	11	2048	sgr1
A_RIGHT	12	4096	sgr1
A_TOP	13	8192	sgr1
A_VERTICAL	14	16384	sgr1
A_ITALIC	15	32768	sitm

For example, on many IBM PC consoles, the underline attribute collides with the foreground color blue and is not available in color mode.

These should have an ncv capability of 2.

SVr4 curses does nothing with ncv, ncurses recognizes it and optimizes the output in favor of colors.

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as pad. Only the first character of the pad string is used. If the terminal does not have a pad character, specify npc. Note that ncurses implements the termcap-compatible PC variable; though the application may set this value to something other than a null, ncurses will test npc first and use napms if the terminal has no pad character.

If the terminal can move up or down half a line, this can be indicated with hu (half-line up) and hd (half-line down). This is primarily use? ful for superscripts and subscripts on hard-copy terminals. If a hard-copy terminal can eject to the next page (form feed), give this as ff (usually control/L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical charac? ters) this can be indicated with the parameterized string rep. The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus, tparm(repeat_char, 'x', 10) is the same as ?xxxxxxxxxx?.

If the terminal has a settable command character, such as the TEKTRONIX 4025, this can be indicated with cmdch. A prototype command character is chosen which is used in all capabilities. This character is given in the cmdch capability to identify it. The following convention is supported on some UNIX systems: The environment is to be searched for a CC variable, and if found, all occurrences of the prototype character are replaced with the character in the environment variable.

Terminal descriptions that do not represent a specific kind of known terminal, such as switch, dialup, patch, and network, should include

the gn (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to virtual terminal descriptions for which the escape sequences are known.)

If the terminal has a ?meta key? which acts as a shift key, setting the 8th bit of any character transmitted, this fact can be indicated with km. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this ?meta mode? on and off, they can be given as smm and rmm.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with lm. A value of lm#0 indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

If the terminal is one of those supported by the UNIX virtual terminal protocol, the terminal number can be given as vt.

Media copy strings which control an auxiliary printer connected to the terminal can be given as mc0: print the contents of the screen, mc4: turn off the printer, and mc5: turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer. It is undefined whether the text is also displayed on the terminal screen when the printer is on. A variation mc5p takes one parameter, and leaves the printer on for as many characters as the value of the param? eter, then turns the printer off. The parameter should not exceed 255. All text, including mc4, is transparently passed to the printer while an mc5p is in effect.

Glitches and Braindamage

Hazeltine terminals, which do not allow ?~? characters to be displayed should indicate hz.

Terminals which ignore a line-feed immediately after an am wrap, such as the Concept and vt100, should indicate xenl.

If el is required to get rid of standout (instead of merely writing normal text on top of it), xhp should be given.

Teleray terminals, where tabs turn all characters moved over to blanks,

should indicate xt (destructive tabs). Note: the variable indicating this is now ?dest_tabs_magic_smso?; in older versions, it was tel? eray_glitch. This glitch is also taken to mean that it is not possible to position the cursor on top of a ?magic cookie?, that to erase stand? out mode it is instead necessary to use delete and insert line. The ncurses implementation ignores this glitch.

The Beehive Superbee, which is unable to correctly transmit the escape or control/C characters, has xsb, indicating that the f1 key is used for escape and f2 for control/C. (Only certain Superbees have this problem, depending on the ROM.) Note that in older terminfo versions, this capability was called ?beehive_glitch?; it is now ?no_esc_ctl_c?.

Other specific terminal problems may be corrected by adding more capa? bilities of the form xx.

Pitfalls of Long Entries

Long terminfo entries are unlikely to be a problem; to date, no entry has even approached terminfo's 4096-byte string-table maximum. Unfor? tunately, the termcap translations are much more strictly limited (to 1023 bytes), thus termcap translations of long terminfo entries can cause problems.

The man pages for 4.3BSD and older versions of tgetent instruct the user to allocate a 1024-byte buffer for the termcap entry. The entry gets null-terminated by the termcap library, so that makes the maximum safe length for a termcap entry 1k-1 (1023) bytes. Depending on what the application and the termcap library being used does, and where in the termcap file the terminal type that tgetent is searching for is, several bad things can happen.

Some termcap libraries print a warning message or exit if they find an entry that's longer than 1023 bytes; others do not; others truncate the entries to 1023 bytes. Some application programs allocate more than the recommended 1K for the termcap entry; others do not.

Each termcap entry has two important sizes associated with it: before ?tc? expansion, and after ?tc? expansion. ?tc? is the capability that tacks on another termcap entry to the end of the current one, to add on

its capabilities. If a termcap entry does not use the ?tc? capability, then of course the two lengths are the same.

The ?before tc expansion? length is the most important one, because it affects more than just users of that particular terminal. This is the length of the entry as it exists in /etc/termcap, minus the backslash-newline pairs, which tgetent strips out while reading it. Some termcap libraries strip off the final newline, too (GNU termcap does not). Now suppose:

? a termcap entry before expansion is more than 1023 bytes long,

? and the application has only allocated a 1k buffer,

? and the termcap library (like the one in BSD/OS 1.1 and GNU) reads the whole entry into the buffer, no matter what its length, to see if it is the entry it wants,

? and tgetent is searching for a terminal type that either is the long entry, appears in the termcap file after the long entry, or does not appear in the file at all (so that tgetent has to search the whole termcap file).

Then tgetent will overwrite memory, perhaps its stack, and probably core dump the program. Programs like telnet are particularly vulnera? ble; modern telnets pass along values like the terminal type automati? cally. The results are almost as undesirable with a termcap library, like SunOS 4.1.3 and Ultrix 4.4, that prints warning messages when it reads an overly long termcap entry. If a termcap library truncates long entries, like OSF/1 3.0, it is immune to dying here but will re? turn incorrect data for the terminal.

The ?after tc expansion? length will have a similar effect to the above, but only for people who actually set TERM to that terminal type, since tgetent only does ?tc? expansion once it is found the terminal type it was looking for, not while searching.

In summary, a termcap entry that is longer than 1023 bytes can cause, on various combinations of termcap libraries and applications, a core dump, warnings, or incorrect operation. If it is too long even before ?tc? expansion, it will have this effect even for users of some other

terminal types and users whose TERM variable does not have a termcap entry.

When in -C (translate to termcap) mode, the ncurses implementation of tic(1M) issues warning messages when the pre-tc length of a termcap translation is too long. The -c (check) option also checks resolved (after tc expansion) lengths.

Binary Compatibility

It is not wise to count on portability of binary terminfo entries be? tween commercial UNIX versions. The problem is that there are at least two versions of terminfo (under HP-UX and AIX) which diverged from Sys? tem V terminfo after SVr1, and have added extension capabilities to the string table that (in the binary format) collide with System V and XSI Curses extensions.

EXTENSIONS

Searching for terminal descriptions in $HOME/.terminfo and TER? MINFO_DIRS is not supported by older implementations.

Some SVr4 curses implementations, and all previous to SVr4, do not in? terpret the %A and %O operators in parameter strings.

SVr4/XPG4 do not specify whether msgr licenses movement while in an al? ternate-character-set mode (such modes may, among other things, map CR and NL to characters that do not trigger local motions). The ncurses implementation ignores msgr in ALTCHARSET mode. This raises the possi? bility that an XPG4 implementation making the opposite interpretation may need terminfo entries made for ncurses to have msgr turned off.

The ncurses library handles insert-character and insert-character modes in a slightly non-standard way to get better update efficiency. See the Insert/Delete Character subsection above.

The parameter substitutions for set_clock and display_clock are not documented in SVr4 or the XSI Curses standard. They are deduced from the documentation for the AT&T 505 terminal.

Be careful assigning the kmous capability. The ncurses library wants to interpret it as KEY_MOUSE, for use by terminals and emulators like xterm that can return mouse-tracking information in the keyboard-input

stream.

X/Open Curses does not mention italics. Portable applications must as? sume that numeric capabilities are signed 16-bit values. This includes the no_color_video (ncv) capability. The 32768 mask value used for italics with ncv can be confused with an absent or cancelled ncv. If italics should work with colors, then the ncv value must be specified, even if it is zero.

Different commercial ports of terminfo and curses support different subsets of the XSI Curses standard and (in some cases) different exten? sion sets. Here is a summary, accurate as of October 1995:

? SVR4, Solaris, ncurses -- These support all SVr4 capabilities.

? SGI -- Supports the SVr4 set, adds one undocumented extended string capability (set_pglen).

? SVr1, Ultrix -- These support a restricted subset of terminfo capa? bilities. The booleans end with xon_xoff; the numerics with width_status_line; and the strings with prtr_non.

? HP/UX -- Supports the SVr1 subset, plus the SVr[234] numerics num_labels, label_height, label_width, plus function keys 11 through 63, plus plab_norm, label_on, and label_off, plus some in? compatible extensions in the string table.

? AIX -- Supports the SVr1 subset, plus function keys 11 through 63, plus a number of incompatible string table extensions.

? OSF -- Supports both the SVr4 set and the AIX extensions.

FILES

/usr/share/terminfo/?/* files containing terminal descriptions

SEE ALSO

infocmp(1M), tabs(1), tic(1M), curses(3X), curs_color(3X), curs_vari? ables(3X), printf(3), term_variables(3X). term(5). user_caps(5).

AUTHORS

Zeyd M. Ben-Halim, Eric S. Raymond, Thomas E. Dickey. Based on pcurses by Pavel Curtis.