NAME

terminfo – terminal capability database

SYNOPSIS

/usr/share/lib/terminfo/?/*

DESCRIPTION

terminfo is a database produced by **tic** that describes the capabilities of devices such as terminals and printers. Devices are described in **terminfo** source files by specifying a set of capabilities, by quantifying certain aspects of the device, and by specifying character sequences that effect particular results. This database is often used by screen oriented applications such as **vi** and **curses** programs, as well as by some system commands such as **ls** and **more**. This usage allows them to work with a variety of devices without changes to the programs.

terminfo source files consist of one or more device descriptions. Each description consists of a header (beginning in column 1) and one or more lines that list the features for that particular device. Every line in a **terminfo** source file must end in a comma (**,**). Every line in a **terminfo** source file except the header must be indented with one or more white spaces (either spaces or tabs).

Entries in **terminfo** source files consist of a number of comma-separated fields. White space after each comma is ignored. Embedded commas must be escaped by using a backslash. The following example shows the format of a **terminfo** source file.

$alias sub 1$ | $alias sub 2$ | ... | $alias sub n$ | *longname*,
<*white space*> **am, lines #24,**
<*white space*> **home=\Eeh,**

The first line, commonly referred to as the header line, must begin in column one and must contain at least two aliases separated by vertical bars. The last field in the header line must be the long name of the device and it may contain any string. Alias names must be unique in the **terminfo** database and they must conform to system file naming conventions (see **tic**(1M)); they cannot, for example, contain white space or slashes.

Every device must be assigned a name, such as "vt100". Device names (except the long name) should be chosen using the following conventions. The name should not contain hyphens because hyphens are reserved for use when adding suffixes that indicate special modes.

These special modes may be modes that the hardware can be in, or user preferences. To assign a special mode to a particular device, append a suffix consisting of a hyphen and an indicator of the mode to the device name. For example, the **-w** suffix means "wide mode"; when specified, it allows for a width of 132 columns instead of the standard 80 columns.

Therefore, if you want to use a "vt100" device set to wide mode, name the device "vt100-w." Use the following suffixes where possible.

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	**5410-w**
-am	With auto. margins (usually default)	**vt100-am**
-nam	Without automatic margins	**vt100-nam**
-*n*	Number of lines on the screen	**2300-40**
-na	No arrow keys (leave them in local)	**c100-na**
-*n*p	Number of pages of memory	**c100-4p**
-rv	Reverse video	**4415-rv**

The **terminfo** reference manual page is organized in two sections:

DEVICE CAPABILITIES and PRINTER CAPABILITIES.

PART 1: DEVICE CAPABILITIES

Capabilities in **terminfo** are of three types: Boolean capabilities (which show that a device has or does not have a particular feature), numeric capabilities (which quantify particular features of a device), and string capabilities (which provide sequences that can be used to perform particular operations on devices).

In the following table, a **Variable** is the name by which a **C** programmer accesses a capability (at the **terminfo** level). A **Capname** is the short name for a capability specified in the **terminfo** source file. It is used by a person updating the source file and by the **tput** command. A **Termcap Code** is a two-letter sequence that corresponds to the **termcap** capability name. (Note that **termcap** is no longer supported.)

Capability names have no real length limit, but an informal limit of five characters has been adopted to keep them short. Whenever possible, capability names are chosen to be the same as or similar to those specified by the ANSI X3.64-1979 standard. Semantics are also intended to match those of the ANSI standard.

All string capabilities listed below may have padding specified, with the exception of those used for input. Input capabilities, listed under the **Strings** section in the following tables, have names beginning with **key_**. The **#i** symbol in the description field of the following tables refers to the *i*th parameter.

Booleans

Variable	Cap-name	Termcap Code	Description
auto_left_margin	**bw**	**bw**	**cub1** wraps from column 0 to last column
auto_right_margin	**am**	**am**	Terminal has automatic margins
back_color_erase	**bce**	**be**	Screen erased with background color
can_change	**ccc**	**cc**	Terminal can re-define existing color
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 resolution
cr_cancels_micro_mode	**crxm**	**YB**	Using **cr** turns off micro mode

dest_tabs_magic_smso	**xt**	**xt**	Destructive tabs, magic **smso** char (t1061)
eat_newline_glitch	**xenl**	**xn**	Newline ignored after 80 columns (Concept)
erase_overstrike	**eo**	**eo**	Can erase overstrikes with a blank
generic_type	**gn**	**gn**	Generic line type (for example, dialup, switch)
get_mouse	**getm**	**Gm**	Curses should get button events
hard_copy	**hc**	**hc**	Hardcopy terminal
hard_cursor	**chts**	**HC**	Cursor is hard to see
has_meta_key	**km**	**km**	Has a meta key (shift, sets parity bit)
has_print_wheel	**daisy**	**YC**	Printer needs operator to change character 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 distinguishes nulls
lpi_changes_res	**lpix**	**YG**	Changing line pitch changes resolution
memory_above	**da**	**da**	Display may be retained above the screen
memory_below	**db**	**db**	Display may be retained below the screen
move_insert_mode	**mir**	**mi**	Safe to move while in insert mode
move_standout_mode	**msgr**	**ms**	Safe to move in standout modes
needs_xon_xoff	**nxon**	**nx**	Padding won't work, xon/xoff required
no_esc_ctlc	**xsb**	**xb**	Beehive (f1=escape, f2=ctrl C)
no_pad_char	**npc**	**NP**	Pad character doesn't exist
non_dest_scroll_region	**ndscr**	**ND**	Scrolling region is nondestructive
non_rev_rmcup	**nrrmc**	**NR**	**smcup** does not reverse **rmcup**
over_strike	**os**	**os**	Terminal overstrikes on hard-copy terminal
prtr_silent	**mc5i**	**5i**	Printer won't 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**	Hazeltine; can't print tilde (˜)
transparent_underline	**ul**	**ul**	Underline character overstrikes
xon_xoff	**xon**	**xo**	Terminal uses xon/xoff handshaking

Numbers

Variable	Cap-name	Termcap Code	Description
bit_image_entwining	**bitwin**	**Yo**	Number of passes for each bit-map row
bit_image_type	**bitype**	**Yp**	Type of bit image device
buffer_capacity	**bufsz**	**Ya**	Number of bytes buffered before printing
buttons	**btns**	**BT**	Number of buttons on the mouse
columns	**cols**	**co**	Number of columns in a line
dot_horz_spacing	**spinh**	**Yc**	Spacing of dots horizontally in dots per inch

dot_vert_spacing	**spinv**	**Yb**	Spacing of pins vertically in pins per inch
init_tabs	**it**	**it**	Tabs initially every # spaces
label_height	**lh**	**lh**	Number of rows in each label
label_width	**lw**	**lw**	Number of columns in each label
lines	**lines**	**li**	Number of lines on a screen or a page
lines_of_memory	**lm**	**lm**	Lines of memory if **> lines**; **0** means varies
max_attributes	**ma**	**ma**	Maximum combined video attributes terminal can display
magic_cookie_glitch	**xmc**	**sg**	Number of blank characters left by **smso** or **rmso**
max_colors	**colors**	**Co**	Maximum number of colors on the screen
max_micro_address	**maddr**	**Yd**	Maximum value in **micro_..._address**
max_micro_jump	**mjump**	**Ye**	Maximum value in **parm_..._micro**
max_pairs	**pairs**	**pa**	Maximum number of color-pairs on the screen
maximum_windows	**wnum**	**MW**	Maximum number of definable windows
micro_char_size	**mcs**	**Yf**	Character step size when in micro mode
micro_line_size	**mls**	**Yg**	Line step size when in micro mode
no_color_video	**ncv**	**NC**	Video attributes that can't be used with colors
num_labels	**nlab**	**Nl**	Number of labels on screen (start at 1)
number_of_pins	**npins**	**Yh**	Number of pins in print-head
output_res_char	**orc**	**Yi**	Horizontal resolution in units per character
output_res_line	**orl**	**Yj**	Vertical resolution in units per line
output_res_horz_inch	**orhi**	**Yk**	Horizontal resolution in units per inch
output_res_vert_inch	**orvi**	**Yl**	Vertical resolution in units per inch
padding_baud_rate	**pb**	**pb**	Lowest baud rate where padding needed
print_rate	**cps**	**Ym**	Print rate in characters per second
virtual_terminal	**vt**	**vt**	Virtual terminal number (system)
wide_char_size	**widcs**	**Yn**	Character step size when in double wide mode
width_status_line	**wsl**	**ws**	Number of columns in status line

Strings

Variable	Cap-name	Termcap Code	Description
acs_chars	**acsc**	**ac**	Graphic charset pairs aAbBcC
alt_scancode_esc	**scesa**	**S8**	Alternate escape for scancode emulation (default is for vt100)
back_tab	**cbt**	**bt**	Back tab
bell	**bel**	**bl**	Audible signal (bell)
bit_image_carriage_return	**bicr**	**Yv**	Move to beginning of same row (use tparm)
bit_image_newline	**binel**	**Zz**	Move to next row of the bit image (use tparm)

bit_image_repeat	**birep**	**Zy**	Repeat bit-image cell #1 #2 times (use tparm)
carriage_return	**cr**	**cr**	Carriage return
change_char_pitch	**cpi**	**ZA**	Change number of characters per inch
change_line_pitch	**lpi**	**ZB**	Change number of lines per inch
change_res_horz	**chr**	**ZC**	Change horizontal resolution
change_res_vert	**cvr**	**ZD**	Change vertical resolution
change_scroll_region	**csr**	**cs**	Change to lines #1 through #2 (vt100)
char_padding	**rmp**	**rP**	Like **ip** but when in replace mode
char_set_names	**csnm**	**Zy**	List of character set names
clear_all_tabs	**tbc**	**ct**	Clear all tab stops
clear_margins	**mgc**	**MC**	Clear all margins (top, bottom, and sides)
clear_screen	**clear**	**cl**	Clear screen and home cursor
clr_bol	**el1**	**cb**	Clear to beginning of line, inclusive
clr_eol	**el**	**ce**	Clear to end of line
clr_eos	**ed**	**cd**	Clear to end of display
code_set_init	**csin**	**ci**	Init sequence for multiple codesets
color_names	**colornm**	**Yw**	Give name for color #1
column_address	**hpa**	**ch**	Horizontal position absolute
command_character	**cmdch**	**CC**	Terminal settable cmd character in prototype
create_window	**cwin**	**CW**	Define win #1 to go from #2,#3 to #4,#5
cursor_address	**cup**	**cm**	Move to row #1 col #2
cursor_down	**cud1**	**do**	Down one line
cursor_home	**home**	**ho**	Home cursor (if no **cup**)
cursor_invisible	**civis**	**vi**	Make cursor invisible
cursor_left	**cub1**	**le**	Move left one space.
cursor_mem_address	**mrcup**	**CM**	Memory relative cursor addressing
cursor_normal	**cnorm**	**ve**	Make cursor appear normal (undo **vs/vi**)
cursor_right	**cuf1**	**nd**	Non-destructive space (cursor or carriage right)
cursor_to_ll	**ll**	**ll**	Last line, first column (if no **cup**)
cursor_up	**cuu1**	**up**	Upline (cursor up)
cursor_visible	**cvvis**	**vs**	Make cursor very visible
define_bit_image_region	**defbi**	**Yx**	Define rectangular bit-image region (use tparm)
define_char	**defc**	**ZE**	Define a character in a character set †
delete_character	**dch1**	**dc**	Delete character
delete_line	**dl1**	**dl**	Delete line
device_type	**devt**	**dv**	Indicate language/codeset support
dial_phone	**dial**	**DI**	Dial phone number #1
dis_status_line	**dsl**	**ds**	Disable status line

display_clock	**dclk**	**DK**	Display time-of-day clock
display_pc_char	**dispc**	**S1**	Display PC character
down_half_line	**hd**	**hd**	Half-line down (forward 1/2 linefeed)
ena_acs	**enacs**	**eA**	Enable alternate character set
end_bit_image_region	**endbi**	**Yy**	End a bit-image region (use tparm)
enter_alt_charset_mode	**smacs**	**as**	Start alternate character set
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 begin programs that use **cup**
enter_delete_mode	**smdc**	**dm**	Delete mode (enter)
enter_dim_mode	**dim**	**mh**	Turn on half-bright mode
enter_doublewide_mode	**swidm**	**ZF**	Enable double wide printing
enter_draft_quality	**sdrfq**	**ZG**	Set draft quality print mode
enter_insert_mode	**smir**	**im**	Insert mode (enter)
enter_italics_mode	**sitm**	**ZH**	Enable italics
enter_leftward_mode	**slm**	**ZI**	Enable leftward carriage motion
enter_micro_mode	**smicm**	**ZJ**	Enable micro motion capabilities
enter_near_letter_quality	**snlq**	**ZK**	Set near-letter quality print
enter_normal_quality	**snrmq**	**ZL**	Set normal quality print
enter_pc_charset_mode	**smpch**	**S2**	Enter PC character display mode
enter_protected_mode	**prot**	**mp**	Turn on protected mode
enter_reverse_mode	**rev**	**mr**	Turn on reverse video mode
enter_scancode_mode	**smsc**	**S4**	Enter PC scancode mode
enter_secure_mode	**invis**	**mk**	Turn on blank mode (characters invisible)
enter_shadow_mode	**sshm**	**ZM**	Enable shadow printing
enter_standout_mode	**smso**	**so**	Begin standout mode
enter_subscript_mode	**ssubm**	**ZN**	Enable subscript printing
enter_superscript_mode	**ssupm**	**ZO**	Enable superscript printing
enter_underline_mode	**smul**	**us**	Start underscore mode
enter_upward_mode	**sum**	**ZP**	Enable upward carriage motion mode
enter_xon_mode	**smxon**	**SX**	Turn on xon/xoff handshaking
erase_chars	**ech**	**ec**	Erase #1 characters
exit_alt_charset_mode	**rmacs**	**ae**	End alternate character set
exit_am_mode	**rmam**	**RA**	Turn off automatic margins
exit_attribute_mode	**sgr0**	**me**	Turn off all attributes
exit_ca_mode	**rmcup**	**te**	String to end programs that use **cup**
exit_delete_mode	**rmdc**	**ed**	End delete mode
exit_doublewide_mode	**rwidm**	**ZQ**	Disable double wide printing
exit_insert_mode	**rmir**	**ei**	End insert mode
exit_italics_mode	**ritm**	**ZR**	Disable italics
exit_leftward_mode	**rlm**	**ZS**	Enable rightward (normal)

			carriage motion
exit_micro_mode	**rmicm**	**ZT**	Disable micro motion capabilities
exit_pc_charset_mode	**rmpch**	**S3**	Disable PC character display mode
exit_scancode_mode	**rmsc**	**S5**	Disable PC scancode mode
exit_shadow_mode	**rshm**	**ZU**	Disable shadow printing
exit_standout_mode	**rmso**	**se**	End standout mode
exit_subscript_mode	**rsubm**	**ZV**	Disable subscript printing
exit_superscript_mode	**rsupm**	**ZW**	Disable superscript printing
exit_underline_mode	**rmul**	**ue**	End underscore mode
exit_upward_mode	**rum**	**ZX**	Enable downward (normal) carriage motion
exit_xon_mode	**rmxon**	**RX**	Turn off xon/xoff handshaking
fixed_pause	**pause**	**PA**	Pause for 2-3 seconds
flash_hook	**hook**	**fh**	Flash the switch hook
flash_screen	**flash**	**vb**	Visible bell (may not move cursor)
form_feed	**ff**	**ff**	Hardcopy terminal page eject
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**	Terminal or printer initialization string
init_2string	**is2**	**is**	Terminal or printer initialization string
init_3string	**is3**	**i3**	Terminal or printer initialization string
init_file	**if**	**if**	Name of initialization file
init_prog	**iprog**	**iP**	Path name of program for initialization
initialize_color	**initc**	**Ic**	Initialize the definition of color
initialize_pair	**initp**	**Ip**	Initialize color-pair
insert_character	**ich1**	**ic**	Insert character
insert_line	**il1**	**al**	Add new blank line
insert_padding	**ip**	**ip**	Insert pad after character inserted

The ''**key_**'' strings are sent by specific keys. The ''**key_**'' descriptions include the macro, defined in **<curses.h>**, for the code returned by the **curses** routine **getch** when the key is pressed (see **curs_getch**(3X)).

Variable	**Cap-name**	**Termcap Code**	**Description**
key_a1	**ka1**	**K1**	**KEY_A1**, upper left of keypad
key_a3	**ka3**	**K3**	**KEY_A3**, upper right of keypad
key_b2	**kb2**	**K2**	**KEY_B2**, center of keypad
key_backspace	**kbs**	**kb**	**KEY_BACKSPACE**, sent by backspace key
key_beg	**kbeg**	**@1**	**KEY_BEG**, sent by beg(inning) key
key_btab	**kcbt**	**kB**	**KEY_BTAB**, sent by back-tab key
key_c1	**kc1**	**K4**	**KEY_C1**, lower left of keypad
key_c3	**kc3**	**K5**	**KEY_C3**, lower right of keypad

key_cancel	**kcan**	**@2**	**KEY_CANCEL**, sent by cancel key
key_catab	**ktbc**	**ka**	**KEY_CATAB**, sent by clear-all-tabs key
key_clear	**kclr**	**kC**	**KEY_CLEAR**, sent by clear-screen or erase key
key_close	**kclo**	**@3**	**KEY_CLOSE**, sent by close key
key_command	**kcmd**	**@4**	**KEY_COMMAND**, sent by cmd (command) key
key_copy	**kcpy**	**@5**	**KEY_COPY**, sent by copy key
key_create	**kcrt**	**@6**	**KEY_CREATE**, sent by create key
key_ctab	**kctab**	**kt**	**KEY_CTAB**, sent by clear-tab key
key_dc	**kdch1**	**kD**	**KEY_DC**, sent by delete-character key
key_dl	**kdl1**	**kL**	**KEY_DL**, sent by delete-line key
key_down	**kcud1**	**kd**	**KEY_DOWN**, sent by terminal down-arrow key
key_eic	**krmir**	**kM**	**KEY_EIC**, sent by **rmir** or **smir** in insert mode
key_end	**kend**	**@7**	**KEY_END**, sent by end key
key_enter	**kent**	**@8**	**KEY_ENTER**, sent by enter/send key
key_eol	**kel**	**kE**	**KEY_EOL**, sent by clear-to-end-of-line key
key_eos	**ked**	**kS**	**KEY_EOS**, sent by clear-to-end-of-screen key
key_exit	**kext**	**@9**	**KEY_EXIT**, sent by exit key
key_f0	**kf0**	**k0**	**KEY_F(0)**, sent by function key f0
key_f1	**kf1**	**k1**	**KEY_F(1)**, sent by function key f1
key_f2	**kf2**	**k2**	**KEY_F(2)**, sent by function key f2
key_f3	**kf3**	**k3**	**KEY_F(3)**, sent by function key f3
key_fB	**kf4**	**k4**	**KEY_F(4)**, sent by function key fB
key_f5	**kf5**	**k5**	**KEY_F(5)**, sent by function key f5
key_f6	**kf6**	**k6**	**KEY_F(6)**, sent by function key f6
key_f7	**kf7**	**k7**	**KEY_F(7)**, sent by function key f7
key_f8	**kf8**	**k8**	**KEY_F(8)**, sent by function key f8
key_f9	**kf9**	**k9**	**KEY_F(9)**, sent by function key f9
key_f10	**kf10**	**k;**	**KEY_F(10)**, sent by function key f10
key_f11	**kf11**	**F1**	**KEY_F(11)**, sent by function key f11
key_f12	**kf12**	**F2**	**KEY_F(12)**, sent by function key f12
key_f13	**kf13**	**F3**	**KEY_F(13)**, sent by function key f13
key_f14	**kf14**	**F4**	**KEY_F(14)**, sent by function key f14
key_f15	**kf15**	**F5**	**KEY_F(15)**, sent by function key f15
key_f16	**kf16**	**F6**	**KEY_F(16)**, sent by function key f16
key_f17	**kf17**	**F7**	**KEY_F(17)**, sent by function key f17
key_f18	**kf18**	**F8**	**KEY_F(18)**, sent by function key f18
key_f19	**kf19**	**F9**	**KEY_F(19)**, sent by function key f19
key_f20	**kf20**	**FA**	**KEY_F(20)**, sent by function key f20
key_f21	**kf21**	**FB**	**KEY_F(21)**, sent by function key f21

key_f22	**kf22**	**FC**	**KEY_F(22)**, sent by function key f22
key_f23	**kf23**	**FD**	**KEY_F(23)**, sent by function key f23
key_f24	**kf24**	**FE**	**KEY_F(24)**, sent by function key f24
key_f25	**kf25**	**FF**	**KEY_F(25)**, sent by function key f25
key_f26	**kf26**	**FG**	**KEY_F(26)**, sent by function key f26
key_f27	**kf27**	**FH**	**KEY_F(27)**, sent by function key f27
key_f28	**kf28**	**FI**	**KEY_F(28)**, sent by function key f28
key_f29	**kf29**	**FJ**	**KEY_F(29)**, sent by function key f29
key_f30	**kf30**	**FK**	**KEY_F(30)**, sent by function key f30
key_f31	**kf31**	**FL**	**KEY_F(31)**, sent by function key f31
key_f32	**kf32**	**FM**	**KEY_F(32)**, sent by function key f32
key_f33	**kf33**	**FN**	**KEY_F(13)**, sent by function key f13
key_f34	**kf34**	**FO**	**KEY_F(34)**, sent by function key f34
key_f35	**kf35**	**FP**	**KEY_F(35)**, sent by function key f35
key_f36	**kf36**	**FQ**	**KEY_F(36)**, sent by function key f36
key_f37	**kf37**	**FR**	**KEY_F(37)**, sent by function key f37
key_f38	**kf38**	**FS**	**KEY_F(38)**, sent by function key f38
key_f39	**kf39**	**FT**	**KEY_F(39)**, sent by function key f39
key_fB0	**kf40**	**FU**	**KEY_F(40)**, sent by function key fB0
key_fB1	**kf41**	**FV**	**KEY_F(41)**, sent by function key fB1
key_fB2	**kf42**	**FW**	**KEY_F(42)**, sent by function key fB2
key_fB3	**kf43**	**FX**	**KEY_F(43)**, sent by function key fB3
key_fB4	**kf44**	**FY**	**KEY_F(44)**, sent by function key fB4
key_fB5	**kf45**	**FZ**	**KEY_F(45)**, sent by function key fB5
key_fB6	**kf46**	**Fa**	**KEY_F(46)**, sent by function key fB6
key_fB7	**kf47**	**Fb**	**KEY_F(47)**, sent by function key fB7
key_fB8	**kf48**	**Fc**	**KEY_F(48)**, sent by function key fB8
key_fB9	**kf49**	**Fd**	**KEY_F(49)**, sent by function key fB9
key_f50	**kf50**	**Fe**	**KEY_F(50)**, sent by function key f50
key_f51	**kf51**	**Ff**	**KEY_F(51)**, sent by function key f51
key_f52	**kf52**	**Fg**	**KEY_F(52)**, sent by function key f52
key_f53	**kf53**	**Fh**	**KEY_F(53)**, sent by function key f53
key_f54	**kf54**	**Fi**	**KEY_F(54)**, sent by function key f54
key_f55	**kf55**	**Fj**	**KEY_F(55)**, sent by function key f55
key_f56	**kf56**	**Fk**	**KEY_F(56)**, sent by function key f56
key_f57	**kf57**	**Fl**	**KEY_F(57)**, sent by function key f57
key_f58	**kf58**	**Fm**	**KEY_F(58)**, sent by function key f58
key_f59	**kf59**	**Fn**	**KEY_F(59)**, sent by function key f59
key_f60	**kf60**	**Fo**	**KEY_F(60)**, sent by function key f60
key_f61	**kf61**	**Fp**	**KEY_F(61)**, sent by function key f61
key_f62	**kf62**	**Fq**	**KEY_F(62)**, sent by function key f62
key_f63	**kf63**	**Fr**	**KEY_F(63)**, sent by function key f63
key_find	**kfnd**	**@0**	**KEY_FIND**, sent by find key
key_help	**khlp**	**%1**	**KEY_HELP**, sent by help key
key_home	**khome**	**kh**	**KEY_HOME**, sent by home key

key_ic	**kich1**	**kI**	**KEY_IC**, sent by ins-char/enter ins-mode key
key_il	**kil1**	**kA**	**KEY_IL**, sent by insert-line key
key_left	**kcub1**	**kl**	**KEY_LEFT**, sent by terminal left-arrow key
key_ll	**kll**	**kH**	**KEY_LL**, sent by home-down key
key_mark	**kmrk**	**%2**	**KEY_MARK**, sent by mark key
key_message	**kmsg**	**%3**	**KEY_MESSAGE**, sent by message key
key_mouse	**kmous**	**Km**	0631, Mouse event has occured
key_move	**kmov**	**%4**	**KEY_MOVE**, sent by move key
key_next	**knxt**	**%5**	**KEY_NEXT**, sent by next-object key
key_npage	**knp**	**kN**	**KEY_NPAGE**, sent by next-page key
key_open	**kopn**	**%6**	**KEY_OPEN**, sent by open key
key_options	**kopt**	**%7**	**KEY_OPTIONS**, sent by options key
key_ppage	**kpp**	**kP**	**KEY_PPAGE**, sent by previous-page key
key_previous	**kprv**	**%8**	**KEY_PREVIOUS**, sent by previous-object key
key_print	**kprt**	**%9**	**KEY_PRINT**, sent by print or copy key
key_redo	**krdo**	**%0**	**KEY_REDO**, sent by redo key
key_reference	**kref**	**&1**	**KEY_REFERENCE**, sent by reference key
key_refresh	**krfr**	**&2**	**KEY_REFRESH**, sent by refresh key
key_replace	**krpl**	**&3**	**KEY_REPLACE**, sent by replace key
key_restart	**krst**	**&4**	**KEY_RESTART**, sent by restart key
key_resume	**kres**	**&5**	**KEY_RESUME**, sent by resume key
key_right	**kcuf1**	**kr**	**KEY_RIGHT**, sent by terminal right-arrow key
key_save	**ksav**	**&6**	**KEY_SAVE**, sent by save key
key_sbeg	**kBEG**	**&9**	**KEY_SBEG**, sent by shifted beginning key
key_scancel	**kCAN**	**&0**	**KEY_SCANCEL**, sent by shifted cancel key
key_scommand	**kCMD**	***1**	**KEY_SCOMMAND**, sent by shifted command key
key_scopy	**kCPY**	***2**	**KEY_SCOPY**, sent by shifted copy key
key_screate	**kCRT**	***3**	**KEY_SCREATE**, sent by shifted create key
key_sdc	**kDC**	***4**	**KEY_SDC**, sent by shifted delete-char key
key_sdl	**kDL**	***5**	**KEY_SDL**, sent by shifted delete-line key
key_select	**kslt**	***6**	**KEY_SELECT**, sent by select key
key_send	**kEND**	***7**	**KEY_SEND**, sent by shifted end key
key_seol	**kEOL**	***8**	**KEY_SEOL**, sent by shifted clear-line key
key_sexit	**kEXT**	***9**	**KEY_SEXIT**, sent by shifted exit key
key_sf	**kind**	**kF**	**KEY_SF**, sent by scroll-forward/down key

key_sfind	**kFND**	***0**	**KEY_SFIND**, sent by shifted find key
key_shelp	**kHLP**	**#1**	**KEY_SHELP**, sent by shifted help key
key_shome	**kHOM**	**#2**	**KEY_SHOME**, sent by shifted home key
key_sic	**kIC**	**#3**	**KEY_SIC**, sent by shifted input key
key_sleft	**kLFT**	**#4**	**KEY_SLEFT**, sent by shifted left-arrow key
key_smessage	**kMSG**	**%a**	**KEY_SMESSAGE**, sent by shifted message key
key_smove	**kMOV**	**%b**	**KEY_SMOVE**, sent by shifted move key
key_snext	**kNXT**	**%c**	**KEY_SNEXT**, sent by shifted next key
key_soptions	**kOPT**	**%d**	**KEY_SOPTIONS**, sent by shifted options key
key_sprevious	**kPRV**	**%e**	**KEY_SPREVIOUS**, sent by shifted prev key
key_sprint	**kPRT**	**%f**	**KEY_SPRINT**, sent by shifted print key
key_sr	**kri**	**kR**	**KEY_SR**, sent by scroll-backward/up key
key_sredo	**kRDO**	**%g**	**KEY_SREDO**, sent by shifted redo key
key_sreplace	**kRPL**	**%h**	**KEY_SREPLACE**, sent by shifted replace key
key_sright	**kRIT**	**%i**	**KEY_SRIGHT**, sent by shifted right-arrow key
key_srsume	**kRES**	**%j**	**KEY_SRSUME**, sent by shifted resume key
key_ssave	**kSAV**	**!1**	**KEY_SSAVE**, sent by shifted save key
key_ssuspend	**kSPD**	**!2**	**KEY_SSUSPEND**, sent by shifted suspend key
key_stab	**khts**	**kT**	**KEY_STAB**, sent by set-tab key
key_sundo	**kUND**	**!3**	**KEY_SUNDO**, sent by shifted undo key
key_suspend	**kspd**	**&7**	**KEY_SUSPEND**, sent by suspend key
key_undo	**kund**	**&8**	**KEY_UNDO**, sent by undo key
key_up	**kcuu1**	**ku**	**KEY_UP**, sent by terminal up-arrow key
keypad_local	**rmkx**	**ke**	Out of ‘‘keypad-transmit’’ mode
keypad_xmit	**smkx**	**ks**	Put terminal in ‘‘keypad-transmit’’ mode
lab_f0	**lf0**	**l0**	Labels on function key f0 if not f0
lab_f1	**lf1**	**l1**	Labels on function key f1 if not f1
lab_f2	**lf2**	**l2**	Labels on function key f2 if not f2
lab_f3	**lf3**	**l3**	Labels on function key f3 if not f3
lab_fB	**lfB**	**l4**	Labels on function key fB if not fB
lab_f5	**lf5**	**l5**	Labels on function key f5 if not f5
lab_f6	**lf6**	**l6**	Labels on function key f6 if not f6
lab_f7	**lf7**	**l7**	Labels on function key f7 if not f7
lab_f8	**lf8**	**l8**	Labels on function key f8 if not f8
lab_f9	**lf9**	**l9**	Labels on function key f9 if not f9

lab_f10	**lf10**	**la**	Labels on function key f10 if not f10
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)
micro_column_address	**mhpa**	**ZY**	Like **column_address** for micro adjustment
micro_down	**mcud1**	**ZZ**	Like **cursor_down** for micro adjustment
micro_left	**mcub1**	**Za**	Like **cursor_left** for micro adjustment
micro_right	**mcuf1**	**Zb**	Like **cursor_right** for micro adjustment
micro_row_address	**mvpa**	**Zc**	Like **row_address** for micro adjustment
micro_up	**mcuu1**	**Zd**	Like **cursor_up** for micro adjustment
mouse_info	**minfo**	**Mi**	Mouse status information
newline	**nel**	**nw**	Newline (behaves like **cr** followed by **lf**)
order_of_pins	**porder**	**Ze**	Matches software bits to print-head pins
orig_colors	**oc**	**oc**	Set all color(-pair)s to the original ones
orig_pair	**op**	**op**	Set default color-pair to the original one
pad_char	**pad**	**pc**	Pad character (rather than null)
parm_dch	**dch**	**DC**	Delete #1 chars
parm_delete_line	**dl**	**DL**	Delete #1 lines
parm_down_cursor	**cud**	**DO**	Move down #1 lines.
parm_down_micro	**mcud**	**Zf**	Like **parm_down_cursor** for micro adjust.
parm_ich	**ich**	**IC**	Insert #1 blank chars
parm_index	**indn**	**SF**	Scroll forward #1 lines.
parm_insert_line	**il**	**AL**	Add #1 new blank lines
parm_left_cursor	**cub**	**LE**	Move cursor left #1 spaces
parm_left_micro	**mcub**	**Zg**	Like **parm_left_cursor** for micro adjust.
parm_right_cursor	**cuf**	**RI**	Move right #1 spaces.
parm_right_micro	**mcuf**	**Zh**	Like **parm_right_cursor** for micro adjust.
parm_rindex	**rin**	**SR**	Scroll backward #1 lines.
parm_up_cursor	**cuu**	**UP**	Move cursor up #1 lines.
parm_up_micro	**mcuu**	**Zi**	Like **parm_up_cursor** for micro adjust.
pc_term_options	**pctrm**	**S6**	PC terminal options
pkey_key	**pfkey**	**pk**	Prog funct key #1 to type string #2
pkey_local	**pfloc**	**pl**	Prog funct key #1 to execute string #2
pkey_plab	**pfxl**	**xl**	Prog key #1 to xmit string #2 and show string #3
pkey_xmit	**pfx**	**px**	Prog funct key #1 to xmit string #2
plab_norm	**pln**	**pn**	Prog label #1 to show string #2

print_screen	**mc0**	**ps**	Print contents of the screen
prtr_non	**mc5p**	**pO**	Turn on the printer for #1 bytes
prtr_off	**mc4**	**pf**	Turn off the printer
prtr_on	**mc5**	**po**	Turn on the printer
pulse	**pulse**	**PU**	Select pulse dialing
quick_dial	**qdial**	**QD**	Dial phone number #1, without progress detection
remove_clock	**rmclk**	**RC**	Remove time-of-day clock
repeat_char	**rep**	**rp**	Repeat char #1 #2 times
req_for_input	**rfi**	**RF**	Send next input char (for ptys)
req_mouse_pos	**reqmp**	**RQ**	Request mouse position report
reset_1string	**rs1**	**r1**	Reset terminal completely to sane modes
reset_2string	**rs2**	**r2**	Reset terminal completely to sane modes
reset_3string	**rs3**	**r3**	Reset terminal completely to sane modes
reset_file	**rf**	**rf**	Name of file containing reset string
restore_cursor	**rc**	**rc**	Restore cursor to position of last sc
row_address	**vpa**	**cv**	Vertical position absolute
save_cursor	**sc**	**sc**	Save cursor position
scancode_escape	**scesc**	**S7**	Escape for scancode emulation
scroll_forward	**ind**	**sf**	Scroll text up
scroll_reverse	**ri**	**sr**	Scroll text down
select_char_set	**scs**	**Zj**	Select character set
set0_des_seq	**s0ds**	**s0**	Shift into codeset 0 (EUC set 0, ASCII)
set1_des_seq	**s1ds**	**s1**	Shift into codeset 1
set2_des_seq	**s2ds**	**s2**	Shift into codeset 2
set3_des_seq	**s3ds**	**s3**	Shift into codeset 3 attributes #1-#6
set_a_background	**setab**	**AB**	Set background color using ANSI escape
set_a_foreground	**setaf**	**AF**	Set foreground color using ANSI escape
set_attributes	**sgr**	**sa**	Define the video attributes #1-#9
set_background	**setb**	**Sb**	Set current background color
set_bottom_margin	**smgb**	**Zk**	Set bottom margin at current line
set_bottom_margin_parm	**smgbp**	**Zl**	Set bottom margin at line #1 or #2 lines from bottom
set_clock	**sclk**	**SC**	Set time-of-day clock
set_color_band	**setcolor**	**Yz**	Change to ribbon color #1
set_color_pair	**scp**	**sp**	Set current color-pair
set_foreground	**setf**	**Sf**	Set current foreground color1
set_left_margin	**smgl**	**ML**	Set left margin at current line
set_left_margin_parm	**smglp**	**Zm**	Set left (right) margin at column #1 (#2)
set_lr_margin	**smglr**	**ML**	Sets both left and right margins
set_page_length	**slines**	**YZ**	Set page length to #1 lines (use tparm) of an inch
set_right_margin	**smgr**	**MR**	Set right margin at current column
set_right_margin_parm	**smgrp**	**Zn**	Set right margin at column #1

set_tab	**hts**	**st**	Set a tab in all rows, current column
set_tb_margin	**smgtb**	**MT**	Sets both top and bottom margins
set_top_margin	**smgt**	**Zo**	Set top margin at current line
set_top_margin_parm	**smgtp**	**Zp**	Set top (bottom) margin at line #1 (#2)
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 definition of a character set
stop_bit_image	**rbim**	**Zs**	End printing bit image graphics
stop_char_set_def	**rcsd**	**Zt**	End definition of a character set
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 chars causes **cr**
to_status_line	**tsl**	**ts**	Go to status line, col #1
tone	**tone**	**TO**	Select touch tone dialing
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
underline_char	**uc**	**uc**	Underscore one char and move past it
up_half_line	**hu**	**hu**	Half-line up (reverse 1/2 linefeed)
wait_tone	**wait**	**WA**	Wait for dial tone
xoff_character	**xoffc**	**XF**	X-off character
xon_character	**xonc**	**XN**	X-on character
zero_motion	**zerom**	**Zx**	No motion for the subsequent character

Sample Entry

The following entry, which describes the AT&T 610 terminal, is among the more complex entries in the **terminfo** file as of this writing.

```
610 | 610bct | ATT610 | att610 | AT&T 610; 80 column; 98key keyboard
    am, eslok, hs, mir, msgr, xenl, xon,
    cols#80, it#8, lh#2, lines#24, lw#8, nlab#8, wsl#80,
    acsc=``aaffggjjkkllmmnnooppqqrrssttuuvvwwxxyyzz{{||}}~~,
    bel=^G, blink=\E[5m, bold=\E[1m, cbt=\E[Z,
    civis=\E[?25l, clear=\E[H\E[J, cnorm=\E[?25h\E[?12l,
    cr=\r, csr=\E[%i%p1%d;%p2%dr, cub=\E[%p1%dD, cub1=\b,
    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,
    cvvis=\E[?12;25h, dch=\E[%p1%dP, dch1=\E[P, dim=\E[2m,
```

```
    dl=\E[%p1%dM, dl1=\E[M, ed=\E[J, el=\E[K, el1=\E[1K,
    flash=\E[?5h$<200>\E[?5l, fsl=\E8, home=\E[H, ht=\t,
    ich=\E[%p1%d@, il=\E[%p1%dL, il1=\E[L, ind=\ED, .ind=\ED$<9>,
    invis=\E[8m,
    is1=\E[8;0 | \E[?3;4;5;13;15l\E[13;20l\E[?7h\E[12h\E(B\E)0,
    is2=\E[0m^O, is3=\E(B\E)0, kLFT=\E[\s@, kRIT=\E[\sA,
    kbs=^H, kcbt=\E[Z, kclr=\E[2J, kcub1=\E[D, kcud1=\E[B,
    kcuf1=\E[C, kcuu1=\E[A, kf1=\EOc, kf10=\ENp,
    kf11=\ENq, kf12=\ENr, kf13=\ENs, kf14=\ENt, kf2=\EOd,
    kf3=\EOe, kf4=\EOf, kf5=\EOg, kf6=\EOh, kf7=\EOi,
    kf8=\EOj, kf9=\ENo, khome=\E[H, kind=\E[S, kri=\E[T,
    ll=\E[24H, mc4=\E[?4i, mc5=\E[?5i, nel=\EE,
    pfxl=\E[%p1%d;%p2%l%02dq%?%p1%{9}%<%t\s\s\sF%p1%1d\s\s\s\s\s
\s\s\s\s\s\s%;%p2%s,
    pln=\E[%p1%d;0;0;0q%p2%:-16.16s, rc=\E8, rev=\E[7m,
    ri=\EM, rmacs=^O, rmir=\E[4l, rmln=\E[2p, rmso=\E[m,
    rmul=\E[m, rs2=\Ec\E[?3l, sc=\E7,
    sgr=\E[0%?%p6%t;1%;%?%p5%t;2%;%?%p2%t;4%;%?%p4%t;5%;
%?%p3%p1% | %t;7%;%?%p7%t;8%;m%?%p9%t^N%e^O%;,
    sgr0=\E[m^O, smacs=^N, smir=\E[4h, smln=\E[p,
    smso=\E[7m, smul=\E[4m, tsl=\E7\E[25;%i%p1%dx,
```

Types of Capabilities in the Sample Entry

The sample entry shows the formats for the three types of **terminfo** capabilities listed: Boolean, numeric, and string. All capabilities specified in the **terminfo** source file must be followed by commas, including the last capability in the source file. In **terminfo** source files, capabilities are referenced by their capability names (as shown in the previous tables).

Boolean capabilities are specified simply by their comma separated cap names.

Numeric capabilities are followed by the character '#' and then a positive integer value. Thus, in the sample, **cols** (which shows the number of columns available on a device) is assigned the value **80** for the AT&T 610. (Values for numeric capabilities may be specified in decimal, octal, or hexadecimal, using normal C programming language conventions.)

Finally, string-valued capabilities such as **el** (clear to end of line sequence) are listed by a two- to five-character capname, an '=', and a string ended by the next occurrence of a comma. A delay in milliseconds may appear anywhere in such a capability, preceded by **$** and enclosed in angle brackets, as in **el=\EK$<3>**. Padding characters are supplied by **tput**. The delay can be any of the following: a number, a number followed by an asterisk, such as **5***, a number followed by a slash, such as **5/**, or a number followed by both, such as **5*/**. A '*' shows 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 characters, the factor is still the number of lines affected. This is always 1 unless the device has **in** and the software uses it.) When a '*' is specified, it is sometimes useful to give a delay of the form **3.5** to specify a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.)

A '/' indicates that the padding is mandatory. If a device has **xon** defined, the padding information is advisory and will only be used for cost estimates or when the device is in raw mode. Mandatory padding will be transmitted regardless of the setting of **xon**. If padding (whether advisory or mandatory) is specified for **bel** or **flash**, however, it will always be used, regardless of whether **xon** is specified.

terminfo offers notation for encoding special characters. 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** give a newline, linefeed, return, tab, backspace, formfeed, and space, respectively. Other escapes include: **\^** for caret (^); **** for backslash (\); **\,** for comma (,); **\:** for colon (:); and **\0** for null. (**\0** will actually produce **\200**, which does not terminate a string but behaves as a null character on most devices, providing CS7 is specified. (See **stty**(1)). Finally, characters may be given as three octal digits after a backslash (for example, \123).

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. Note that capabilities are defined in a left-to-right order and, therefore, a prior definition will override a later definition.

Preparing Descriptions

The most effective way to prepare a device description is by imitating the description of a similar device in **terminfo** and building up a description gradually, using partial descriptions with **vi** to check that they are correct. Be aware that a very unusual device may expose deficiencies in the ability of the **terminfo** file to describe it or the inability of **vi** to work with that device. To test a new device description, set the environment variable **TERMINFO** to the pathname of a directory containing the compiled description you are working on and programs will look there rather than in **/usr/share/lib/terminfo**. To get the padding for insert-line correct (if the device manufacturer did not document it) a severe test is to comment out **xon**, edit a large file at 9600 baud with **vi**, delete 16 or so lines from the middle of the screen, and then press the **u** key several times quickly. If the display is corrupted, more padding is usually needed. A similar test can be used for insert-character.

Section 1-1: Basic Capabilities

The number of columns on each line for the device is given by the **cols** numeric capability. If the device has a screen, then the number of lines on the screen is given by the **lines** capability. If the device 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 overstrikes (rather than clearing a position when a character is

struck over) then it should have the **os** capability. If the device is a printing terminal, with no soft copy unit, specify both **hc** and **os**. If there is a way to move the cursor to the left edge of the current row, specify this as **cr**.
(Normally this will be carriage return, control M.) If there is a way to produce an audible signal (such as a bell or a beep), specify it as **bel**. If, like most devices, the device uses the xon-xoff flow-control protocol, specify **xon**.

If there is a way to move the cursor one position to the left (such as backspace), that capability should be given as **cub1**. Similarly, sequences to move to the right, up, and down should be given as **cuf1**, **cuu1**, and **cud1**, respectively. These local cursor motions must not alter the text they pass over; for example, you would not normally use ''**cuf1**=\s'' 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 screen terminal. Programs should never attempt to backspace around the left edge, unless **bw** is specified, and should never attempt to go up locally off the top. To scroll text up, a program goes to the bottom left corner of the screen and sends 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**. These versions have the same semantics as **ind** and **ri**, except that they take one parameter and scroll the number of lines specified by that parameter. 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. Backward motion from the left edge of the screen is possible only when **bw** is specified. In this case, **cub1** 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 device has switch selectable automatic margins, **am** should be specified in the **terminfo** source file. In this case, initialization strings should turn on this option, if possible. If the device has a command that moves to the first column 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 device 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 hardcopy and screen terminals. Thus the AT&T 5320 hardcopy terminal is described as follows:

```
5320|att5320|AT&T 5320 hardcopy terminal,
    am, hc, os,
    cols#132,
    bel=^G, cr=\r, cub1=\b, cnd1=\n,
```

dch1=\E[P, dl1=\E[M,
ind=\n,

while the Lear Siegler ADM–3 is described as

adm3 | lsi adm3,
am, bel=^G, clear=^Z, cols#80, cr=^M, cub1=^H,
cud1=^J, ind=^J, lines#24,

Section 1-2: Parameterized Strings

Cursor addressing and other strings requiring parameters are described by a parameterized string capability, with **printf**-like escapes (%*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 the stack in the manner of Reverse Polish Notation (postfix). Typically a sequence will push one of the parameters onto the stack and then print it in some format. Often more complex operations are necessary. Operations are in postfix form with the operands in the usual order. That is, to subtract 5 from the first parameter, one would use **%p1%{5}%–**.

The % encodings have the following meanings:

%% outputs '%'

%**[[:]***flags*][*width*[.*precision*]][**doxXs**]
as in **printf**, flags are **[–+#]** and space

%c print pop gives %c

%p[1-9]
push *i*th parm

%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

%'*c*' push char constant *c*

%{*nn*} push decimal constant *nn*

%l push strlen(pop)

%+ %– %* %/ %m
arithmetic (%**m** is mod): push(pop $integer sub 2$ op pop $integer sub 1$)

%& %| %^
bit operations: push(pop $integer sub 2$ op pop $integer sub 1$)

%= %> %<
logical operations: push(pop $integer sub 2$ op pop $integer sub 1$)

%A %O
logical operations: and, or

%! %~ unary operations: push(op pop)

%i (for ANSI terminals) add 1 to first parm, if one parm present, or first two parms, if more than one parm present

%? *expr* **%t** *thenpart* **%e** *elsepart* **%;**
if-then-else, **%e** *elsepart* is optional; else-if's are possible ala Algol 68: **%?** $\mathbf{c_1}$ **%t** $\mathbf{b_1}$ **%e** $\mathbf{c_2}$ **%t** $\mathbf{b_2}$ **%e** $\mathbf{c_3}$ **%t** $\mathbf{b_3}$ **%e** $\mathbf{c_4}$ **%t** $\mathbf{b_4}$ **%e** $\mathbf{b_5}$**%;**
c_i are conditions, b_i are bodies.

If the ''–'' flag is used with ''%[**doxXs**]'', then a colon (**:**) must be placed between the ''%'' and the ''–'' to differentiate the flag from the binary ''%–'' operator, for example ''**%:–16.16s**''.

Consider the Hewlett-Packard 2645, 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 zero-padded as two digits. Thus its **cup** capability is:

cup=\E&a%p2%2.2dc%p1%2.2dY$<6>

The Micro-Term 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**''. Devices that 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%'\s'%+%c%p2%'\s'%+%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.

Section 1-3: 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 corner of the screen, not of memory. (Thus, the **\EH** sequence on Hewlett-Packard terminals cannot be used for **home** without losing some of the other features on the terminal.)

If the device has row or column absolute-cursor addressing, these can be given as single parameter capabilities **hpa** (horizontal position absolute) and **vpa** (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to **cup**. If there are parameterized local motions (for example, 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 device does not have **cup**, such as the Tektronix 4025.

If the device 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, such as the Concept, with more than one page of memory. If the device has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the device 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 **rmcup**), specify **nrrmc**.

Section 1-4: 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 available.)

Section 1-5: Insert/Delete Line

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 destructive 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 this command — the **sc** and **rc** (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using **ri** or **ind** on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

To determine whether a terminal has destructive scrolling regions or non-destructive scrolling regions, create a scrolling region in the middle of the screen, place data on the bottom line of the scrolling region, move the cursor to the top line of the scrolling region, and do a reverse index (**ri**) followed by a delete line (**dl1**) or index (**ind**). If the data that was originally on the bottom line of the scrolling region was restored into the scrolling region by the **dl1** or **ind**, then the terminal has non-destructive scrolling regions. Otherwise, it has destructive scrolling regions. Do not specify **csr** if the terminal has non-destructive scrolling regions, unless **ind**, **ri**, **indn**, **rin**, **dl**, and **dl1** all simulate destructive scrolling.

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 a full screen may bring non-blank lines up from below or that scrolling back with **ri** may bring down non-blank lines.

Section 1-6: Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character operations 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 second type of terminal, and should give the capability **in**, which stands for ''insert null.'' While these are two logically separate attributes (one line versus multiline 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 that 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 sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ich1**; terminals that send a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to **ich1**. Do not give both unless the terminal actually requires both to be used in combination.) If post-insert padding is needed, give this as a number of milliseconds padding 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 parameter, *n*, will insert *n* blanks.

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 (for example, 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.

Section 1-7: Highlighting, Underlining, and Visible Bells

Your device may have one or more kinds of display attributes that allow you to highlight selected characters when they appear on the screen. The following display modes (shown with the names by which they are set) may be available: a blinking screen (**blink**), bold or extra-bright characters (**bold**), dim or half-bright characters (**dim**), blanking or invisible text (**invis**), protected text (**prot**), a reverse-video screen (**rev**), and an alternate character set (**smacs** to enter this mode and **rmacs** to exit it). (If a command is necessary before you can enter alternate character set mode, give the sequence in **enacs** or "enable alternate-character-set" mode.) Turning on any of these modes singly may or may not turn off other modes.

sgr0 should be used to turn off all video enhancement capabilities. It should always be specified because it represents the only way to turn off some capabilities, such as **dim** or **blink**.

You should choose one display method as *standout mode* and use it to highlight error messages and other kinds of text to which you want to draw attention. Choose a form of display that provides strong contrast but that is easy on the eyes. (We recommend reverse-video plus half-bright 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.

Sequences to begin underlining and end underlining can be specified as **smul** and **rmul ,** respectively. If the device has a sequence to underline the current character and to move the cursor one space to the right (such as the Micro-Term MIME), this sequence can be specified as **uc**.

Terminals with the ''magic cookie'' glitch (**xmc)** deposit special ''cookies'' when they receive mode-setting sequences, which affect the display algorithm rather than having extra bits for each character. Some terminals, such as the Hewlett-Packard 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. A good flash can be done by changing the screen into reverse video, pad for 200 ms, then return the screen to normal video.

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**. The boolean **chts** should also be given. 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 either of these modes.

If your terminal generates underlined characters by using the underline character (with no special sequences needed) even though it does not otherwise overstrike characters, then you should specify the capability **ul**. For devices on which 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 specifying **eo**.

If there is a sequence to set arbitrary combinations of modes, this should be given as **sgr** (set attributes), taking nine parameters. Each parameter is either **0** or non-zero, as the corresponding attribute is on or off. The nine parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, alternate character set. Not all modes need to be supported by **sgr**; only those for which corresponding separate attribute commands exist should be supported. For example, let's assume that the terminal in question needs the following escape sequences to turn on various modes.

tparm parameter	attribute	escape sequence
	none	**\E[0m**
p1	standout	**\E[0;4;7m**
p2	underline	**\E[0;3m**
p3	reverse	**\E[0;4m**
p4	blink	**\E[0;5m**
p5	dim	**\E[0;7m**
p6	bold	**\E[0;3;4m**
p7	invis	**\E[0;8m**
p8	protect	not available
p9	altcharset	**^O** (off) **^N** (on)

Note that each escape sequence requires a **0** to turn off other modes before turning on its own mode. Also note that, as suggested above, *standout* is set up to be the combination of *reverse* and *dim*. Also, because this terminal has no *bold* mode, *bold* is set up as the combination of *reverse* and *underline*. In addition, to allow combinations, such as *underline+blink*, the sequence to use would be **\E[0;3;5m**. The terminal doesn't have *protect* mode, either, but that cannot be simulated in any way, so **p8** is ignored. The *altcharset* mode is different in that it is either **^O** or **^N**, depending on whether it is off or on. If all modes were to be turned on, the sequence would be **\E[0;3;4;5;7;8m^N**.

Now look at when different sequences are output. For example, **;3** is output when either **p2** or **p6** is true, that is, if either *underline* or *bold* modes are turned on.
Writing out the above sequences, along with their dependencies, gives the following:

sequence	when to output	terminfo translation
\E[0	always	**\E[0**
;3	if **p2** or **p6**	**%?%p2%p6%|%t;3%;**
;4	if **p1** or **p3** or **p6**	**%?%p1%p3%|%p6%|%t;4%;**
;5	if **p4**	**%?%p4%t;5%;**
;7	if **p1** or **p5**	**%?%p1%p5%|%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%?%p2%p6%|%t;3%;%?%p1%p3%|%p6%
|%t;4%;%?%p5%t;5%;%?%p1%p5%
|%t;7%;%?%p7%t;8%;m%?%p9%t^N%e^O%;,

Remember that **sgr** and **sgr0** must always be specified.

Section 1-8: Keypad

If the device has a keypad that transmits sequences when the keys are pressed, this information can also be specified. Note that it is not possible to handle devices where the keypad only works in local (this applies, for example, to the unshifted Hewlett-Packard

2621 keys). If the keypad can be set to transmit or not transmit, specify these sequences as **smkx** and **rmkx**. Otherwise the keypad is assumed to always transmit.

The sequences sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kcub1, kcuf1, kcuu1, kcud1,** and **khome**, respectively. If there are function keys such as f0, f1, ..., f63, the sequences they send can be specified as **kf0, kf1, ..., kf63**. If the first 11 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 directional pad are needed. Further keys are defined above in the capabilities list.

Strings to program function keys can be specified as **pfkey**, **pfloc**, and **pfx**. A string to program screen labels should be specified as **pln**. Each of these strings takes two parameters: a function key identifier and a string to program it with. **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 local mode; 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 normally output after one or more **pln** sequences to make sure that the change becomes visible.

Section 1-9: Tabs and Initialization

If the device has hardware tabs, the command to advance to the next tab stop can be given as **ht** (usually control I). A ''backtab'' command that moves leftward to the next tab stop can be given as **cbt**. By convention, if tty modes show that tabs are being expanded by the computer rather than being sent to the device, programs should not use **ht** or **cbt** (even if they are present) because the user may not have the tab stops properly set. If the device has hardware tabs that are initially set every *n* spaces when the device is powered up, the numeric parameter **it** is given, showing the number of spaces the tabs are set to. This is normally used by **tput init** (see **tput**(1)) to determine whether to set the mode for hardware tab expansion and whether to set the tab stops. If the device has tab stops that can be saved in nonvolatile memory, the **terminfo** description can assume that they are properly set. 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).

Other capabilities include: **is1**, **is2**, and **is3**, initialization strings for the device; **iprog**, the path name of a program to be run to initialize the device; and **if**, the name of a file containing long initialization strings. These strings are expected to set the device into modes consistent with the rest of the **terminfo** description. They must be sent to the device each time the user logs in and be output in the following order: run the program **iprog**; output **is1**; output **is2**; set the margins using **mgc**, **smgl** and **smgr**; set the tabs using **tbc** and **hts**; print the file **if**; and finally output **is3**. This is usually done using the **init** option of **tput**.

Most initialization is done with **is2**. Special device modes can be set up without duplicating strings by putting the common sequences in **is2** and special cases in **is1** and **is3**. Sequences that do a reset from a totally unknown state can be given as **rs1**, **rs2**, **rf**, and **rs3**, analogous to **is1**, **is2**, **is3**, and **if**. (The method using files, **if** and **rf**, is used for a few terminals, from **/usr/share/lib/tabset/***; however, the recommended method is to use the initialization and reset strings.) These strings are output by **tput** reset, 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 a terminal into 80-column mode would normally be part of **is2**, but on some terminals it causes an annoying glitch on the screen and is not normally needed because the terminal is usually already in 80-column mode.

If a more complex sequence is needed to set the tabs than can be described by using **tbc** and **hts**, the sequence can be placed in **is2** or **if**.

Any margin can be cleared with **mgc**. (For instructions on how to specify commands to set and clear margins, see "Margins" below under "PRINTER CAPABILITIES.")

Section 1-10: Delays

Certain capabilities control padding in the **tty** driver. These are primarily needed by hard-copy terminals, and are used by **tput init** to set tty modes appropriately. Delays embedded in the capabilities **cr**, **ind**, **cub1**, **ff**, and **tab** can be used to set the appropriate delay bits to be set in the tty driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**.

Section 1-11: Status Lines

If the terminal has an extra ''status line'' that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, into which one can cursor address normally (such as the Heathkit h19's 25th line, or the 24th line of a VT100 which is set to a 23-line scrolling region), the capability **hs** should be given. Special strings that go to a given column of the status line and return from the status line can be given as **tsl** and **fsl**. (**fsl** must leave the cursor position in the same place it was before **tsl**. If necessary, the **sc** and **rc** strings can be included in **tsl** and **fsl** to get this effect.) The capability **tsl** takes one parameter, which is the column number of the status line the cursor is to be moved to.

If escape sequences and other special commands, such as tab, work while in the status line, the flag **eslok** can be given. A string which turns off the status line (or otherwise erases its contents) should be given as **dsl**. If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**. The status line is normally assumed to be the same width as the rest of the screen, for example, **cols**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded) the width, in columns, can be indicated with the numeric parameter **wsl**.

Section 1-12: Line Graphics

If the device has a line drawing alternate character set, the mapping of glyph to character would be given in **acsc**. The definition of this string is based on the alternate character set used in the DEC VT100 terminal, extended slightly with some characters from the AT&T 4410v1 terminal.

glyph name	vt100+ character
arrow pointing right	**+**
arrow pointing left	**,**
arrow pointing down	**.**
solid square block	**0**
lantern symbol	**I**
arrow pointing up	**–**
diamond	**‘**
checker board (stipple)	**a**
degree symbol	**f**
plus/minus	**g**
board of squares	**h**
lower right corner	**j**
upper right corner	**k**
upper left corner	**l**
lower left corner	**m**
plus	**n**
scan line 1	**o**
horizontal line	**q**
scan line 9	**s**
left tee (├)	**t**
right tee (┤)	**u**
bottom tee (┴)	**v**
top tee (┬)	**w**
vertical line	**x**
bullet	**~**

The best way to describe a new device's line graphics set is to add a third column to the above table with the characters for the new device that produce the appropriate glyph when the device is in the alternate character set mode. For example,

glyph name	vt100+ char	new tty char
upper left corner	**l**	**R**
lower left corner	**m**	**F**
upper right corner	**k**	**T**
lower right corner	**j**	**G**
horizontal line	**q**	**,**
vertical line	**x**	**.**

Now write down the characters left to right, as in ''**acsc=lRmFkTjGq\,x.**''.

In addition, **terminfo** allows you to define multiple character sets. See Section 2-5 for details.

Section 1-13: Color Manipulation

Let us define two methods of color manipulation: the Tektronix method and the HP method. The Tektronix method uses a set of N predefined colors (usually 8) from which a user can select "current" foreground and background colors. Thus a terminal can support up to N colors mixed into N∗N color-pairs to be displayed on the screen at the same time. When using an HP method the user cannot define the foreground independently of the background, or vice-versa. Instead, the user must define an entire color-pair at once. Up to M color-pairs, made from 2∗M different colors, can be defined this way. Most existing color terminals belong to one of these two classes of terminals.

The numeric variables **colors** and **pairs** define the number of colors and color-pairs that can be displayed on the screen at the same time. If a terminal can change the definition of a color (for example, the Tektronix 4100 and 4200 series terminals), this should be specified with **ccc** (can change color). To change the definition of a color (Tektronix 4200 method), use **initc** (initialize color). It requires four arguments: color number (ranging from 0 to **colors**–1) and three RGB (red, green, and blue) values or three HLS colors (Hue, Lightness, Saturation). Ranges of RGB and HLS values are terminal dependent.

Tektronix 4100 series terminals only use HLS color notation. For such terminals (or dual-mode terminals to be operated in HLS mode) one must define a boolean variable **hls**; that would instruct the **curses init_color** routine to convert its RGB arguments to HLS before sending them to the terminal. The last three arguments to the **initc** string would then be HLS values.

If a terminal can change the definitions of colors, but uses a color notation different from RGB and HLS, a mapping to either RGB or HLS must be developed.

To set current foreground or background to a given color, use **setaf** (set ANSI foreground) and **setab** (set ANSI background). They require one parameter: the number of the color. To initialize a color-pair (HP method), use **initp** (initialize pair). It requires seven parameters: the number of a color-pair (range=0 to **pairs**−1), and six RGB values: three for the foreground followed by three for the background. (Each of these groups of three should be in the order RGB.) When **initc** or **initp** are used, RGB or HLS arguments should be in the order "red, green, blue" or "hue, lightness, saturation"), respectively. To make a color-pair current, use **scp** (set color-pair). It takes one parameter, the number of a color-pair.

Some terminals (for example, most color terminal emulators for PCs) erase areas of the screen with current background color. In such cases, **bce** (background color erase) should be defined. The variable **op** (original pair) contains a sequence for setting the foreground and the background colors to what they were at the terminal start-up time. Similarly, **oc** (original colors) contains a control sequence for setting all colors (for the Tektronix method) or color-pairs (for the HP method) to the values they had at the terminal start-up time.

Some color terminals substitute color for video attributes. Such video attributes should not be combined with colors. Information about these video attributes should be packed into the **ncv** (no color video) variable. There is a one-to-one correspondence between the nine least significant bits of that variable and the video attributes. The following table depicts this correspondence.

Attribute	Bit Position	Decimal Value
A_STANDOUT	0	1
A_UNDERLINE	1	2
A_REVERSE	2	4
A_BLINK	3	8
A_DIM	4	16
A_BOLD	5	32
A_INVIS	6	64
A_PROTECT	7	128
A_ALTCHARSET	8	256

When a particular video attribute should not be used with colors, the corresponding **ncv** bit should be set to 1; otherwise it should be set to zero. To determine the information to pack into the **ncv** variable, you must add together the decimal values corresponding to those attributes that cannot coexist with colors. For example, if the terminal uses colors to simulate reverse video (bit number 2 and decimal value 4) and bold (bit number 5 and decimal value 32), the resulting value for **ncv** will be 36 (4 + 32).

Section 1-14: 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**.

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 useful for superscripts and subscripts on hardcopy terminals. If a hardcopy 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 characters) 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 systems: If the environment variable **CC** exists, all occurrences of the prototype character are replaced with the character in **CC**.

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 is one of those supported by the system virtual terminal protocol, the terminal number can be given as **vt**. A line-turn-around sequence to be transmitted before doing reads should be specified in **rfi**.

If the device uses xon/xoff handshaking for flow control, give **xon**. Padding information should still be included so that routines can make better decisions about costs, but actual pad characters will not be transmitted. Sequences to turn on and off xon/xoff handshaking may be given in **smxon** and **rmxon**. If the characters used for handshaking are not **^S** and **^Q**, they may be specified with **xonc** and **xoffc**.

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.

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.

A variation, **mc5p**, takes one parameter, and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. If the text is not displayed on the terminal screen when the printer is on, specify **mc5i** (silent printer). All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Section 1-15: Special Cases

The working model used by **terminfo** fits most terminals reasonably well. However, some terminals do not completely match that model, requiring special support by **terminfo**. These are not meant to be construed as deficiencies in the terminals; they are just differences between the working model and the actual hardware. They may be unusual devices or, for some reason, do not have all the features of the **terminfo** model implemented.

Terminals that cannot display tilde (˜) characters, such as certain Hazeltine terminals, should indicate **hz**.

Terminals that ignore a linefeed immediately after an **am** wrap, such as the Concept 100, should indicate **xenl**. Those terminals whose cursor remains on the right-most column until another character has been received, rather than wrapping immediately upon receiving the right-most character, such as the VT100, should also indicate **xenl**.

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

Those Teleray terminals whose tabs turn all characters moved over to blanks, should indicate **xt** (destructive tabs). This capability is also taken to mean that it is not possible to position the cursor on top of a ''magic cookie.'' Therefore, to erase standout mode, it is necessary, instead, to use delete and insert line.

Those Beehive Superbee terminals which do not transmit the escape or control–C characters, should specify **xsb**, indicating that the f1 key is to be used for escape and the f2 key for control C.

Section 1-16: Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **use** can be given with the name of the similar terminal. The capabilities given before **use** override those in the terminal type invoked by **use**. A capability can be canceled by placing *xx*@ to the left of the capability definition, where *xx* is the capability. For example, the entry

```
att4424-2|Teletype 4424 in display function group ii,
    rev@, sgr@, smul@, use=att4424,
```

defines an AT&T 4424 terminal that does not have the **rev**, **sgr**, and **smul** capabilities, and hence cannot do highlighting. This is useful for different modes for a terminal, or for different user preferences. More than one **use** capability may be given.

PART 2: PRINTER CAPABILITIES

The **terminfo** database allows you to define capabilities of printers as well as terminals. To find out what capabilities are available for printers as well as for terminals, see the two lists under "DEVICE CAPABILITIES" that list capabilities by variable and by capability name.

Section 2-1: Rounding Values

Because parameterized string capabilities work only with integer values, we recommend that **terminfo** designers create strings that expect numeric values that have been rounded. Application designers should note this and should always round values to the nearest integer before using them with a parameterized string capability.

Section 2-2: Printer Resolution

A printer's resolution is defined to be the smallest spacing of characters it can achieve. In general printers have independent resolution horizontally and vertically. Thus the vertical resolution of a printer can be determined by measuring the smallest achievable distance between consecutive printing baselines, while the horizontal resolution can be determined by measuring the smallest achievable distance between the left-most edges of consecutive printed, identical, characters.

All printers are assumed to be capable of printing with a uniform horizontal and vertical resolution. The view of printing that **terminfo** currently presents is one of printing inside a uniform matrix: All characters are printed at fixed positions relative to each ''cell'' in the matrix; furthermore, each cell has the same size given by the smallest horizontal and vertical step sizes dictated by the resolution. (The cell size can be changed as will be seen later.)

Many printers are capable of ''proportional printing,'' where the horizontal spacing depends on the size of the character last printed. **terminfo** does not make use of this capability, although it does provide enough capability definitions to allow an application to simulate proportional printing.

A printer must not only be able to print characters as close together as the horizontal and vertical resolutions suggest, but also of ''moving'' to a position an integral multiple of the smallest distance away from a previous position. Thus printed characters can be spaced apart a distance that is an integral multiple of the smallest distance, up to the length or width of a single page.

Some printers can have different resolutions depending on different ''modes.'' In ''normal mode,'' the existing **terminfo** capabilities are assumed to work on columns and lines, just like a video terminal. Thus the old **lines** capability would give the length of a page in lines, and the **cols** capability would give the width of a page in columns. In ''micro mode,'' many **terminfo** capabilities work on increments of lines and columns. With some printers the micro mode may be concomitant with normal mode, so that all the capabilities work at the same time.

Section 2-3: Specifying Printer Resolution

The printing resolution of a printer is given in several ways. Each specifies the resolution as the number of smallest steps per distance:

Specification of Printer Resolution

Characteristic	Number of Smallest Steps
orhi	Steps per inch horizontally
orvi	Steps per inch vertically
orc	Steps per column
orl	Steps per line

When printing in normal mode, each character printed causes movement to the next column, except in special cases described later; the distance moved is the same as the per-column resolution. Some printers cause an automatic movement to the next line when a character is printed in the rightmost position; the distance moved vertically is the same as the per-line resolution.

When printing in micro mode, these distances can be different, and may be zero for some printers.

Specification of Printer Resolution

Automatic Motion after Printing

Normal Mode:	
orc	Steps moved horizontally
orl	Steps moved vertically
Micro Mode:	
mcs	Steps moved horizontally
mls	Steps moved vertically

Some printers are capable of printing wide characters. The distance moved when a wide character is printed in normal mode may be different from when a regular width character is printed. The distance moved when a wide character is printed in micro mode may also be different from when a regular character is printed in micro mode, but the differences are assumed to be related: If the distance moved for a regular character is the same whether in normal mode or micro mode (**mcs=orc**), then the distance moved for a wide character is also the same whether in normal mode or micro mode. This doesn't mean the normal character distance is necessarily the same as the wide character distance, just that the distances don't change with a change in normal to micro mode. However, if the distance moved for a regular character is different in micro mode from the distance moved in normal mode (**mcs<orc**), the micro mode distance is assumed to be the same for a wide character printed in micro mode, as the table below shows.

Specification of Printer Resolution
Automatic Motion after Printing Wide Character

Normal Mode or Micro Mode (**mcs** = **orc**):	
widcs	Steps moved horizontally
Micro Mode (**mcs** < **orc**):	
mcs	Steps moved horizontally

There may be control sequences to change the number of columns per inch (the character pitch) and to change the number of lines per inch (the line pitch). If these are used, the resolution of the printer changes, but the type of change depends on the printer:

Specification of Printer Resolution
Changing the Character/Line Pitches

cpi	Change character pitch
cpix	If set, **cpi** changes **orhi**, otherwise changes **orc**
lpi	Change line pitch
lpix	If set, **lpi** changes **orvi**, otherwise changes **orl**
chr	Change steps per column
cvr	Change steps per line

The **cpi** and **lpi** string capabilities are each used with a single argument, the pitch in columns (or characters) and lines per inch, respectively. The **chr** and **cvr** string capabilities are each used with a single argument, the number of steps per column and line, respectively.

Using any of the control sequences in these strings will imply a change in some of the values of **orc**, **orhi**, **orl**, and **orvi**. Also, the distance moved when a wide character is printed, **widcs**, changes in relation to **orc**. The distance moved when a character is printed in micro mode, **mcs**, changes similarly, with one exception: if the distance is 0 or 1, then no change is assumed (see items marked with † in the following table).

Programs that use **cpi**, **lpi**, **chr**, or **cvr** should recalculate the printer resolution (and should recalculate other values— see "Effect of Changing Printing Resolution" under "Dot-Mapped Graphics").

Specification of Printer Resolution
Effects of Changing the Character/Line Pitches

Before	*After*
Using **cpi** with **cpix** clear:	
$bold orhi ′$	**orhi**
$bold orc ′$	$bold orc = bold orhi over V sub italic cpi$
Using **cpi** with **cpix** set:	
$bold orhi ′$	$bold orhi = bold orc cdot V sub italic cpi$
$bold orc ′$	$bold orc$
Using **lpi** with **lpix** clear:	
$bold orvi ′$	$bold orvi$
$bold orl ′$	$bold orl = bold orvi over V sub italic lpi$
Using **lpi** with **lpix** set:	
$bold orvi ′$	$bold orvi = bold orl cdot V sub italic lpi$
$bold orl ′$	$bold orl$
Using **chr:**	
$bold orhi ′$	$bold orhi$
$bold orc ′$	$V sub italic chr$
Using **cvr:**	
$bold orvi ′$	$bold orvi$
$bold orl ′$	$V sub italic cvr$
Using **cpi** or **chr:**	
$bold widcs ′$	$bold widcs = bold {widcs ′} bold orc over { bold {orc ′} }$
$bold mcs ′$	$bold mcs = bold {mcs ′} bold orc over { bold {orc ′} }$

$V sub italic cpi$, $V sub italic lpi$, $V sub italic chr$, and $V sub italic cvr$ are the arguments used with **cpi**, **lpi**, **chr**, and **cvr**, respectively. The prime marks (′) indicate the old values.

Section 2-4: Capabilities that Cause Movement

In the following descriptions, ''movement'' refers to the motion of the ''current position.'' With video terminals this would be the cursor; with some printers this is the carriage position. Other printers have different equivalents. In general, the current position is where a character would be displayed if printed.

terminfo has string capabilities for control sequences that cause movement a number of full columns or lines. It also has equivalent string capabilities for control sequences that cause movement a number of smallest steps.

String Capabilities for Motion

mcub1	Move 1 step left
mcuf1	Move 1 step right
mcuu1	Move 1 step up
mcud1	Move 1 step down
mcub	Move *N* steps left
mcuf	Move *N* steps right
mcuu	Move *N* steps up
mcud	Move *N* steps down
mhpa	Move *N* steps from the left
mvpa	Move *N* steps from the top

The latter six strings are each used with a single argument, *N*.

Sometimes the motion is limited to less than the width or length of a page. Also, some printers don't accept absolute motion to the left of the current position. **terminfo** has capabilities for specifying these limits.

Limits to Motion

mjump	Limit on use of **mcub1**, **mcuf1**, **mcuu1**, **mcud1**
maddr	Limit on use of **mhpa**, **mvpa**
xhpa	If set, **hpa** and **mhpa** can't move left
xvpa	If set, **vpa** and **mvpa** can't move up

If a printer needs to be in a ''micro mode'' for the motion capabilities described above to work, there are string capabilities defined to contain the control sequence to enter and exit this mode. A boolean is available for those printers where using a carriage return causes an automatic return to normal mode.

Entering/Exiting Micro Mode

smicm	Enter micro mode
rmicm	Exit micro mode
crxm	Using **cr** exits micro mode

The movement made when a character is printed in the rightmost position varies among printers. Some make no movement, some move to the beginning of the next line, others move to the beginning of the same line. **terminfo** has boolean capabilities for describing all three cases.

What Happens After Character
Printed in Rightmost Position

sam	Automatic move to beginning of same line

Some printers can be put in a mode where the normal direction of motion is reversed. This mode can be especially useful when there are no capabilities for leftward or upward motion, because those capabilities can be built from the motion reversal capability and the rightward or downward motion capabilities. It is best to leave it up to an application to build the leftward or upward capabilities, though, and not enter them in the **terminfo** database. This allows several reverse motions to be strung together without intervening wasted steps that leave and reenter reverse mode.

Entering/Exiting Reverse Modes	
slm	Reverse sense of horizontal motions
rlm	Restore sense of horizontal motions
sum	Reverse sense of vertical motions
rum	Restore sense of vertical motions
While sense of horizontal motions reversed:	
mcub1	Move 1 step right
mcuf1	Move 1 step left
mcub	Move *N* steps right
mcuf	Move *N* steps left
cub1	Move 1 column right
cuf1	Move 1 column left
cub	Move *N* columns right
cuf	Move *N* columns left
While sense of vertical motions reversed:	
mcuu1	Move 1 step down
mcud1	Move 1 step up
mcuu	Move *N* steps down
mcud	Move *N* steps up
cuu1	Move 1 line down
cud1	Move 1 line up
cuu	Move *N* lines down
cud	Move *N* lines up

The reverse motion modes should not affect the **mvpa** and **mhpa** absolute motion capabilities. The reverse vertical motion mode should, however, also reverse the action of the line ''wrapping'' that occurs when a character is printed in the right-most position. Thus printers that have the standard **terminfo** capability **am** defined should experience motion to the beginning of the previous line when a character is printed in the right-most position under reverse vertical motion mode.

The action when any other motion capabilities are used in reverse motion modes is not defined; thus, programs must exit reverse motion modes before using other motion capabilities.

Two miscellaneous capabilities complete the list of new motion capabilities. One of these is needed for printers that move the current position to the beginning of a line when certain control characters, such as ''line-feed'' or ''form-feed,'' are used. The other is used for the capability of suspending the motion that normally occurs after printing a character.

Miscellaneous Motion Strings

docr	List of control characters causing **cr**
zerom	Prevent auto motion after printing next single character

Margins

terminfo provides two strings for setting margins on terminals: one for the left and one for the right margin. Printers, however, have two additional margins, for the top and bottom margins of each page. Furthermore, some printers require not using motion strings to move the current position to a margin and then fixing the margin there, but require the specification of where a margin should be regardless of the current position. Therefore **terminfo** offers six additional strings for defining margins with printers.

Setting Margins

smgl	Set left margin at current column
smgr	Set right margin at current column
smgb	Set bottom margin at current line
smgt	Set top margin at current line
smgbp	Set bottom margin at line *N*
smglp	Set left margin at column *N*
smgrp	Set right margin at column *N*
smgtp	Set top margin at line *N*

The last four strings are used with one or more arguments that give the position of the margin or margins to set. If both of **smglp** and **smgrp** are set, each is used with a single argument, *N,* that gives the column number of the left and right margin, respectively. If both of **smgtp** and **smgbp** are set, each is used to set the top and bottom margin, respectively: **smgtp** is used with a single argument, *N,* the line number of the top margin; however, **smgbp** is used with two arguments, *N* and *M,* that give the line number of the bottom margin, the first counting from the top of the page and the second counting from the bottom. This accommodates the two styles of specifying the bottom margin in different manufacturers' printers. When coding a **terminfo** entry for a printer that has a settable bottom margin, only the first or second parameter should be used, depending on the printer. When writing an application that uses **smgbp** to set the bottom margin, both arguments must be given.

If only one of **smglp** and **smgrp** is set, then it is used with two arguments, the column number of the left and right margins, in that order. Likewise, if only one of **smgtp** and **smgbp** is set, then it is used with two arguments that give the top and bottom margins, in that order, counting from the top of the page. Thus when coding a **terminfo** entry for a printer that requires setting both left and right or top and bottom margins simultaneously, only one of **smglp** and **smgrp** or **smgtp** and **smgbp** should be defined; the other should be left blank. When writing an application that uses these string capabilities, the pairs should be first checked to see if each in the pair is set or only one is set, and should then be used accordingly.

In counting lines or columns, line zero is the top line and column zero is the left-most column. A zero value for the second argument with **smgbp** means the bottom line of the page.

All margins can be cleared with **mgc**.

Shadows, Italics, Wide Characters

Five new sets of strings describe the capabilities printers have of enhancing printed text.

Enhanced Printing

sshm	Enter shadow-printing mode
rshm	Exit shadow-printing mode
sitm	Enter italicizing mode
ritm	Exit italicizing mode
swidm	Enter wide character mode
rwidm	Exit wide character mode
ssupm	Enter superscript mode
rsupm	Exit superscript mode
supcs	List of characters available as superscripts
ssubm	Enter subscript mode
rsubm	Exit subscript mode
subcs	List of characters available as subscripts

If a printer requires the **sshm** control sequence before every character to be shadow-printed, the **rshm** string is left blank. Thus programs that find a control sequence in **sshm** but none in **rshm** should use the **sshm** control sequence before every character to be shadow-printed; otherwise, the **sshm** control sequence should be used once before the set of characters to be shadow-printed, followed
by **rshm**. The same is also true of each of the **sitm**/**ritm**, **swidm**/**rwidm**, **ssupm**/**rsupm**, and **ssubm**/ **rsubm** pairs.

Note that **terminfo** also has a capability for printing emboldened text (**bold**). While shadow printing and emboldened printing are similar in that they ''darken'' the text, many printers produce these two types of print in slightly different ways. Generally, emboldened printing is done by overstriking the same character one or more times. Shadow printing likewise usually involves overstriking, but with a slight movement up and/or to the side so that the character is ''fatter.''

It is assumed that enhanced printing modes are independent modes, so that it would be possible, for instance, to shadow print italicized subscripts.

As mentioned earlier, the amount of motion automatically made after printing a wide character should be given in **widcs**.

If only a subset of the printable ASCII characters can be printed as superscripts or subscripts, they should be listed in **supcs** or **subcs** strings, respectively. If the **ssupm** or **ssubm** strings contain control sequences, but the corresponding **supcs** or **subcs** strings are empty, it is assumed that all printable ASCII characters are available as superscripts or subscripts.

Automatic motion made after printing a superscript or subscript is assumed to be the same as for regular characters. Thus, for example, printing any of the following three examples will result in equivalent motion:

Bi $\mathbf{B_i}$ $\mathbf{B^i}$

Note that the existing **msgr** boolean capability describes whether motion control sequences can be used while in ''standout mode.'' This capability is extended to cover the enhanced printing modes added here. **msgr** should be set for those printers that accept any motion control sequences without affecting shadow, italicized, widened, superscript, or subscript printing. Conversely, if **msgr** is not set, a program should end these modes before attempting any motion.

Section 2-5: Alternate Character Sets

In addition to allowing you to define line graphics (described in Section 1-12), **terminfo** lets you define alternate character sets. The following capabilities cover printers and terminals with multiple selectable or definable character sets.

Alternate Character Sets

scs	Select character set *N*
scsd	Start definition of character set *N*, *M* characters
defc	Define character *A*, *B* dots wide, descender *D*
rcsd	End definition of character set *N*
csnm	List of character set names
daisy	Printer has manually changed print-wheels

The **scs**, **rcsd**, and **csnm** strings are used with a single argument, *N*, a number from 0 to 63 that identifies the character set. The **scsd** string is also used with the argument *N* and another, *M*, that gives the number of characters in the set. The **defc** string is used with three arguments: *A* gives the ASCII code representation for the character, *B* gives the width of the character in dots, and *D* is zero or one depending on whether the character is a ''descender'' or not. The **defc** string is also followed by a string of ''image-data'' bytes that describe how the character looks (see below).

Character set 0 is the default character set present after the printer has been initialized. Not every printer has 64 character sets, of course; using **scs** with an argument that doesn't select an available character set should cause a null result from **tparm**.

If a character set has to be defined before it can be used, the **scsd** control sequence is to be used before defining the character set, and the **rcsd** is to be used after. They should also cause a null result from **tparm** when used with an argument *N* that doesn't apply. If a character set still has to be selected after being defined, the **scs** control sequence should follow the **rcsd** control sequence. By examining the results of using each of the **scs**, **scsd**, and **rcsd** strings with a character set number in a call to **tparm**, a program can determine which of the three are needed.

Between use of the **scsd** and **rcsd** strings, the **defc** string should be used to define each character. To print any character on printers covered by **terminfo**, the ASCII code is sent to the printer. This is true for characters in an alternate set as well as ''normal'' characters. Thus the definition of a character includes the ASCII code that represents it. In addition, the width of the character in dots is given, along with an indication of whether the character should descend below the print line (such as the lower case letter ''g'' in most character sets). The width of the character in dots also indicates the number of image-data bytes that will follow the **defc** string. These image-data bytes indicate where in a dot-matrix pattern ink should be applied to ''draw'' the character; the number of these bytes and their form are defined below under ''Dot-Mapped Graphics.''

It's easiest for the creator of **terminfo** entries to refer to each character set by number; however, these numbers will be meaningless to the application developer. The **csnm** string alleviates this problem by providing names for each number.

When used with a character set number in a call to **tparm**, the **csnm** string will produce the equivalent name. These names should be used as a reference only. No naming convention is implied, although anyone who creates a **terminfo** entry for a printer should use names consistent with the names found in user documents for the printer. Application developers should allow a user to specify a character set by number (leaving it up to the user to examine the **csnm** string to determine the correct number), or by name, where the application examines the **csnm** string to determine the corresponding character set number.

These capabilities are likely to be used only with dot-matrix printers. If they are not available, the strings should not be defined. For printers that have manually changed print-wheels or font cartridges, the boolean **daisy** is set.

Section 2-6: Dot-Matrix Graphics

Dot-matrix printers typically have the capability of reproducing ''raster-graphics'' images. Three new numeric capabilities and three new string capabilities can help a program draw raster-graphics images independent of the type of dot-matrix printer or the number of pins or dots the printer can handle at one time.

Dot-Matrix Graphics

npins	Number of pins, *N*, in print-head
spinv	Spacing of pins vertically in pins per inch
spinh	Spacing of dots horizontally in dots per inch
porder	Matches software bits to print-head pins
sbim	Start printing bit image graphics, *B* bits wide
rbim	End printing bit image graphics

The **sbim** sring is used with a single argument, *B*, the width of the image in dots.

The model of dot-matrix or raster-graphics that **terminfo** presents is similar to the technique used for most dot-matrix printers: each pass of the printer's print-head is assumed to produce a dot-matrix that is *N* dots high and *B* dots wide. This is typically a wide, squat, rectangle of dots. The height of this rectangle in dots will vary from one printer to the next; this is given in the **npins** numeric capability. The size of the rectangle in fractions of an inch will also vary; it can be deduced from the **spinv** and **spinh** numeric capabilities.
With these three values an application can divide a complete raster-graphics image into several horizontal strips, perhaps interpolating to account for different dot spacing vertically and horizontally.

The **sbim** and **rbim** strings are used to start and end a dot-matrix image, respectively. The **sbim** string is used with a single argument that gives the width of the dot-matrix in dots. A sequence of ''image-data bytes'' are sent to the printer after the **sbim** string and before the **rbim** string. The number of bytes is a integral multiple of the width of the dot-matrix; the multiple and the form of each byte is determined by the **porder** string as described below.

The **porder** string is a comma separated list of pin numbers optionally followed by an numerical offset. The offset, if given, is separated from the list with a semicolon. The position of each pin number in the list corresponds to a bit in an 8-bit data byte. The pins are numbered consecutively from 1 to **npins**, with 1 being the top pin. Note that the term ''pin'' is used loosely here; ''ink-jet'' dot-matrix printers don't have pins, but can be considered to have an equivalent method of applying a single dot of ink to paper. The bit positions in **porder** are in groups of 8, with the first position in each group the most significant bit and the last position the least significant bit. An application produces 8-bit bytes in the order of the groups in **porder**.

An application computes the ''image-data bytes'' from the internal image, mapping vertical dot positions in each print-head pass into 8-bit bytes, using a 1 bit where ink should be applied and 0 where no ink should be applied. This can be reversed (0 bit for ink, 1 bit for no ink) by giving a negative pin number. If a position is skipped in **porder**, a 0 bit is used. If a position has a lower case 'x' instead of a pin number, a 1 bit is used in the skipped position. For consistency, a lower case 'o' can be used to represent a 0 filled, skipped bit. There must be a multiple of 8 bit positions used or skipped in **porder**; if not, 0 bits are used to fill the last byte in the least significant bits. The offset, if given, is added to each data byte; the offset can be negative.

Some examples may help clarify the use of the **porder** string. The AT&T 470, AT&T 475 and C.Itoh 8510 printers provide eight pins for graphics. The pins are identified top to bottom by the 8 bits in a byte, from least significant to most. The **porder** strings for these printers would be **8,7,6,5,4,3,2,1**. The AT&T 478 and AT&T 479 printers also provide eight pins for graphics. However, the pins are identified in the reverse order. The **porder** strings for these printers would be **1,2,3,4,5,6,7,8**. The AT&T 5310, AT&T 5320, DEC LA100, and DEC LN03 printers provide six pins for graphics. The pins are identified top to bottom by the decimal values 1, 2, 4, 8, 16 and 32. These correspond to the low six bits in an 8-bit byte, although the decimal values are further offset by the value 63. The **porder** string for these printers would be **,,6,5,4,3,2,1;63**, or alternately **o,o,6,5,4,3,2,1;63**.

Section 2-7: Effect of Changing Printing Resolution

If the control sequences to change the character pitch or the line pitch are used, the pin or dot spacing may change:

Dot-Matrix Graphics
Changing the Character/Line Pitches

cpi	Change character pitch
cpix	If set, **cpi** changes **spinh**
lpi	Change line pitch
lpix	If set, **lpi** changes **spinv**

Programs that use **cpi** or **lpi** should recalculate the dot spacing:

Dot-Matrix Graphics
Effects of Changing the Character/Line Pitches

Before	*After*
Using **cpi** with **cpix** clear:	
$bold spinh ′$	$bold spinh$
Using **cpi** with **cpix** set:	
$bold spinh ′$	$bold spinh = bold spinh ′ cdot bold orhi over { bold {orhi ′} }$
Using **lpi** with **lpix** clear:	
$bold spinv ′$	$bold spinv$
Using **lpi** with **lpix** set:	
$bold spinv ′$	$bold spinv = bold {spinv ′} cdot bold orhi over { bold {orhi ′}}$
Using **chr:**	
$bold spinh ′$	$bold spinh$
Using **cvr:**	
$bold spinv ′$	$bold spinv$

orhi′ and **orhi** are the values of the horizontal resolution in steps per inch, before using **cpi** and after using **cpi**, respectively. Likewise, **orvi′** and **orvi** are the values of the vertical resolution in steps per inch, before using **lpi** and after using **lpi**, respectively. Thus, the changes in the dots per inch for dot-matrix graphics follow the changes in steps per inch for printer resolution.

Section 2-8: Print Quality

Many dot-matrix printers can alter the dot spacing of printed text to produce near ''letter quality'' printing or ''draft quality'' printing. Usually it is important to be able to choose one or the other because the rate of printing generally falls off as the quality improves. There are three new strings used to describe these capabilities.

Print Quality	
snlq	Set near-letter quality print
snrmq	Set normal quality print
sdrfq	Set draft quality print

The capabilities are listed in decreasing levels of quality. If a printer doesn't have all three levels, one or two of the strings should be left blank as appropriate.

Section 2-9: Printing Rate and Buffer Size

Because there is no standard protocol that can be used to keep a program synchronized with a printer, and because modern printers can buffer data before printing it, a program generally cannot determine at any time what has been printed. Two new numeric capabilities can help a program estimate what has been printed.

Print Rate/Buffer Size	
cps	Nominal print rate in characters per second
bufsz	Buffer capacity in characters

cps is the nominal or average rate at which the printer prints characters; if this value is not given, the rate should be estimated at one-tenth the prevailing baud rate. **bufsz** is the maximum number of subsequent characters buffered before the guaranteed printing of an earlier character, assuming proper flow control has been used. If this value is not given it is assumed that the printer does not buffer characters, but prints them as they are received.

As an example, if a printer has a 1000-character buffer, then sending the letter ''a'' followed by 1000 additional characters is guaranteed to cause the letter ''a'' to print. If the same printer prints at the rate of 100 characters per second, then it should take 10 seconds to print all the characters in the buffer, less if the buffer is not full. By keeping track of the characters sent to a printer, and knowing the print rate and buffer size, a program can synchronize itself with the printer.

Note that most printer manufacturers advertise the maximum print rate, not the nominal print rate. A good way to get a value to put in for **cps** is to generate a few pages of text, count the number of printable characters, and then see how long it takes to print the text.

Applications that use these values should recognize the variability in the print rate. Straight text, in short lines, with no embedded control sequences will probably print at close to the advertised print rate and probably faster than the rate in **cps**. Graphics data with a lot of control sequences, or very long lines of text, will print at well below the advertised rate and below the rate in **cps**. If the application is using **cps** to decide how long it should take a printer to print a block of text, the application should pad the estimate. If the application is using **cps** to decide how much text has already been printed, it should shrink the estimate. The application will thus err in favor of the user, who wants, above all, to see all the output in its correct place.

FILES

/usr/share/lib/terminfo/?/*	compiled terminal description database
/usr/share/lib/.COREterm/?/*	subset of compiled terminal description database
/usr/share/lib/tabset/*	tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs)

SEE ALSO

ls(1), **pg**(1), **stty**(1), **tput**(1), **tty**(1), **vi**(1), **tic**(1M), **printf**(3S)

NOTES

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in **terminfo** and to build up a description gradually, using partial descriptions with a screen oriented editor, such as **vi**, to check that they are correct. To easily test a new terminal description the environment variable **TERMINFO** can be set to the pathname of a directory containing the compiled description, and programs will look there rather than in **/usr/share/lib/terminfo**.