C Curses

Version 2 Release 1
## Contents

**About This Book** ........................................... xiii
- Typographical conventions .................................... xiii
- Other documents ................................................. xiv
- Where to find more information ................................ xiv

**Summary of Changes** ........................................ xvii

**Chapter 1. The Curses Library** ............................. 1
- Terminology .................................................... 1
- Naming Conventions ............................................. 2
- Structure of a Curses Program ................................ 3
  - Return Values ................................................ 3

**Chapter 2. Initializing Curses** ............................. 5

**Chapter 3. Windows in the Curses Environment** ........... 7
- The Default Window Structure .................................. 7
- The Current Window Structure ................................... 8
- Subwindows ......................................................... 8
- Pads ............................................................... 9

**Chapter 4. Manipulating Window Data with Curses** ....... 11

**Chapter 5. Curses Interfaces** ............................... 13
- addch() .......................................................... 13
  - Name ......................................................... 13
  - Synopsis .................................................... 13
  - Description .................................................. 13
  - Return Value ................................................ 13
  - Errors ....................................................... 13
  - Application Usage ......................................... 13
  - See Also ...................................................... 13
- addchstr() ...................................................... 14
  - Name ......................................................... 14
  - Synopsis .................................................... 14
  - Description .................................................. 14
  - Return Value ................................................ 14
  - Errors ....................................................... 14
  - Application Usage ......................................... 14
  - See Also ...................................................... 14
- addstr() ........................................................ 15
  - Name ......................................................... 15
  - Synopsis .................................................... 15
  - Description .................................................. 15
  - Return Value ................................................ 15
  - Errors ....................................................... 15
  - See Also ...................................................... 15
- addwch() ........................................................ 16
  - Name ......................................................... 16
  - Synopsis .................................................... 16
  - Description .................................................. 16
  - Return Value ................................................ 16
  - See Also ...................................................... 16
- addstr() ........................................................ 17
  - Name ......................................................... 17
  - Synopsis .................................................... 17
  - Description .................................................. 17
  - Return Value ................................................ 17
  - Errors ....................................................... 17
  - See Also ...................................................... 17
- addwchstr() ..................................................... 18
  - Name ......................................................... 18
  - Synopsis .................................................... 18
  - Description .................................................. 18
  - Return Value ................................................ 18
  - Errors ....................................................... 18
  - See Also ...................................................... 18
- attron() ........................................................ 18
  - Name ......................................................... 18
  - Synopsis .................................................... 18
  - Description .................................................. 18
  - Return Value ................................................ 18
  - Errors ....................................................... 18
  - See Also ...................................................... 18
- attroff() ........................................................ 19
  - Name ......................................................... 19
  - Synopsis .................................................... 19
  - Description .................................................. 19
  - Return Value ................................................ 19
  - Errors ....................................................... 19
  - See Also ...................................................... 19
- att_get() ........................................................ 19
  - Name ......................................................... 19
  - Synopsis .................................................... 19
  - Description .................................................. 19
  - Return Value ................................................ 19
  - Errors ....................................................... 19
  - See Also ...................................................... 19
- baudrate() ....................................................... 20
  - Name ......................................................... 20
  - Synopsis .................................................... 20
  - Description .................................................. 20
  - Return Value ................................................ 20
  - Errors ....................................................... 20
  - See Also ...................................................... 20
- beep() ............................................................ 21
  - Name ......................................................... 21
  - Synopsis .................................................... 21
  - Description .................................................. 21
  - Return Value ................................................ 21
  - Errors ....................................................... 21
  - See Also ...................................................... 21
- bkgrnd() .......................................................... 22
  - Name ......................................................... 22
  - Synopsis .................................................... 22
  - Description .................................................. 22
  - Return Value ................................................ 22
  - Errors ....................................................... 22
  - See Also ...................................................... 22
- bkgrnd() .......................................................... 22
  - Name ......................................................... 22
  - Synopsis .................................................... 22
  - Description .................................................. 22
  - Return Value ................................................ 22
  - Errors ....................................................... 22
  - See Also ...................................................... 22
- bkgrnd() .......................................................... 23
  - Name ......................................................... 23
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Return Value</th>
<th>Errors</th>
<th>Application Usage</th>
<th>See Also</th>
</tr>
</thead>
<tbody>
<tr>
<td>def_prog_mode()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delay_output()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delch()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>del_curterm()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deleteln()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delscreen()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delwin()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>derwin()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doupdate()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dupwin()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>echo()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>echochar()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>echo_wchar()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>endwin()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>erase()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>erasechar()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>filter()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td>See Also</td>
<td>Application Usage</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------</td>
<td>--------------</td>
<td>----------</td>
<td>----------</td>
<td>-------------------</td>
</tr>
<tr>
<td>flash()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>flushinp()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getbegyx()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getbgd()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getbkgnd()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getch()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getmaxyx()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getnstr()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getn_wstr()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getparyx()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getstr()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>get_wch()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getyx()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>halfdelay()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>has_colors()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>has_ic()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
instr() ................................ 73
Name ................................ 73
Synopsis ................................ 74
Description ................................ 74
ins_wch() ................................ 74
Name ................................ 74
Synopsis ................................ 74
Description ................................ 74
Return Value ................................ 74
Errors ................................ 74
Application Usage ................................ 74
See Also ................................ 74
ins_wstr() ................................ 74
Name ................................ 74
Synopsis ................................ 74
Description ................................ 75
intrflush() ................................ 75
Name ................................ 75
Synopsis ................................ 75
Description ................................ 75
Return Value ................................ 75
Errors ................................ 75
Application Usage ................................ 75
See Also ................................ 75
in_wch() ................................ 75
Name ................................ 75
Synopsis ................................ 75
Description ................................ 75
Return Value ................................ 76
Errors ................................ 76
See Also ................................ 76
in_wchnstr() ................................ 76
Name ................................ 76
Synopsis ................................ 76
Description ................................ 76
Return Value ................................ 77
Errors ................................ 77
Application Usage ................................ 77
See Also ................................ 77
inwstr() ................................ 77
Name ................................ 77
Synopsis ................................ 77
Description ................................ 77
isendwin() ................................ 77
Name ................................ 77
Synopsis ................................ 77
Description ................................ 77
Return Value ................................ 77
Errors ................................ 77
See Also ................................ 78
is_linetouched() ................................ 78
Name ................................ 78
Synopsis ................................ 78
Description ................................ 78
Return Value ................................ 78
Errors ................................ 78
Application Usage ................................ 79
See Also ................................ 79
keyname() ................................ 79
Name ................................ 79
Synopsis ................................ 79
keypad() ................................ 80
Name ................................ 80
Synopsis ................................ 80
Description ................................ 80
Return Value ................................ 80
Errors ................................ 80
See Also ................................ 80
killchar() ................................ 80
Name ................................ 80
Synopsis ................................ 80
Description ................................ 81
leaveok() ................................ 81
Name ................................ 81
Synopsis ................................ 81
Description ................................ 81
LINES ................................ 81
Name ................................ 81
Synopsis ................................ 81
Description ................................ 81
Return Value ................................ 82
Errors ................................ 82
Application Usage ................................ 82
See Also ................................ 82
longname() ................................ 82
Name ................................ 82
Synopsis ................................ 82
Description ................................ 82
Return Value ................................ 82
Errors ................................ 82
Application Usage ................................ 82
See Also ................................ 82
move() ................................ 83
Name ................................ 83
Synopsis ................................ 83
Description ................................ 83
Return Value ................................ 83
Errors ................................ 83
See Also ................................ 83
mv ................................ 83
Name ................................ 83
Synopsis ................................ 83
Description ................................ 83
mvcur() ................................ 84
Name ................................ 84
Synopsis ................................ 84
Description ................................ 84
Return Value ................................ 85
Errors ................................ 85
Application Usage ................................ 85
See Also ................................ 85
mvderwin() ................................ 85
See Also ................................ 85
<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Synopsis</th>
<th>Description</th>
<th>Return Value</th>
<th>Errors</th>
<th>See Also</th>
<th>Application Usage</th>
<th>See Also</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvprintw()</td>
<td>mvprintw()</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td>See Also</td>
<td></td>
<td></td>
</tr>
<tr>
<td>napms()</td>
<td>napms()</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td>See Also</td>
<td></td>
<td></td>
</tr>
<tr>
<td>newpad()</td>
<td>newpad()</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td>See Also</td>
<td></td>
<td></td>
</tr>
<tr>
<td>newterm()</td>
<td>newterm()</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td>See Also</td>
<td></td>
<td></td>
</tr>
<tr>
<td>newwin()</td>
<td>newwin()</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td>See Also</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nl()</td>
<td>nl()</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td>See Also</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodelay()</td>
<td>nodelay()</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td>See Also</td>
<td></td>
<td></td>
</tr>
<tr>
<td>noqiflush()</td>
<td>noqiflush()</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td>See Also</td>
<td></td>
<td></td>
</tr>
<tr>
<td>notimeout()</td>
<td>notimeout()</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td>See Also</td>
<td></td>
<td></td>
</tr>
<tr>
<td>overlay()</td>
<td>overlay()</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td>See Also</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pair_content()</td>
<td>pair_content()</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td>See Also</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pechochar()</td>
<td>pechochar()</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td>See Also</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pnoutrefresh()</td>
<td>pnoutrefresh()</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td>See Also</td>
<td></td>
<td></td>
</tr>
<tr>
<td>printw()</td>
<td>printw()</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td>See Also</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td>Application Usage</td>
<td>See Also</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>-------------</td>
<td>--------------</td>
<td>--------</td>
<td>-------------------</td>
<td>----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>qiflush()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>raw()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redrawwin()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>refresh()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reset_prog_mode()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>resetty()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>restartterm()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ripoffline()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>savetty()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>scr_dump()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>scrl()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>scrollok()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>setchar()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>setscrreg()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set_term()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set_curterm()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>setupterm()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slk_attroff()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>standend()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>start_color()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stdscr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Name</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td>See Also</td>
<td>Application Usage</td>
<td>See Also</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>----------------</td>
<td>--------------------</td>
<td>--------------</td>
<td>----------</td>
<td>----------</td>
<td>--------------------</td>
<td>----------</td>
</tr>
<tr>
<td>subpad()</td>
<td>Name</td>
<td>Synopsis</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subwin()</td>
<td>Name</td>
<td>Synopsis</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>synck()</td>
<td>Name</td>
<td>Synopsis</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>termattr()</td>
<td>Name</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>termname()</td>
<td>Name</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tgetent()</td>
<td>Name</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tgetflag()</td>
<td>Name</td>
<td>Synopsis</td>
<td>Description</td>
<td>Return Value</td>
<td>Errors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>timeout()</td>
<td>Name</td>
<td>Synopsis</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>touchline()</td>
<td>Name</td>
<td>Synopsis</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tparm()</td>
<td>Name</td>
<td>Synopsis</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tputs()</td>
<td>Name</td>
<td>Synopsis</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
About This Book

This manual describes the curses interface for application programs using the z/OS C language. Readers are expected to be experienced C language programmers and to be familiar with open systems standards or a UNIX operating system. This book also assumes that readers are somewhat familiar with MVS systems and with the information for MVS and its accompanying products. Readers also should have read z/OS Introduction and Release Guide which describes the services and the concepts of. This manual is organized as follows:

- **Chapter 1, “The Curses Library,” on page 1** gives an overview of Curses. It discusses the use of some of the key data types and gives general rules for important common concepts such as characters, renditions and window properties. It contains general rules for the common Curses operations and operating modes. This information is implicitly referenced by the interface definitions in Chapter 2. The chapter explains the system of naming the Curses functions and presents a table of function families. Finally, the chapter contains notes regarding use of macros and restrictions on block-mode terminals.

- **Chapter 5, “Curses Interfaces,” on page 13** defines the Curses functional interfaces.

- **Chapter 6, “Headers,” on page 125** defines the contents of headers, which declare constants, macros and data structures that are needed by programs using the services provided by **Chapter 7, “Terminfo Source Format (ENHANCED CURSES),” on page 141**.

- **Chapter 7, “Terminfo Source Format (ENHANCED CURSES),” on page 141** discusses the terminfo database, which Curses uses to describe terminals. The chapter specifies the source format of a terminfo entry, using a formal grammar, an informal discussion, and an example. Boolean, numeric and string capabilities are presented in tabular form. The remainder of the chapter discusses the use of these capabilities by the writer of a terminfo entry to describe the characteristics of the terminal in use.

- The glossary contains definitions of terms used in this manual.

Typographical conventions

The following typographical conventions are used throughout this document:

- **Bold font** is used in text for options to commands, filenames, keywords, type names, data structures and their members.

- **Italic strings** are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  - Command operands, command option-arguments or variable names, for example, substitutable argument prototypes
  - Environment variables, which are also shown in capitals
  - Utility names
  - External variables, such as errno
  - Functions; these are shown as follows: name(); names without parentheses are C external variables, C function family names, utility names, command operands or command option-arguments.

- **Normal font** is used for the names of constants and literals.

- The notation `<file.h>` indicates a header file.
• Names surrounded by braces, for example, \{ARG_MAX\}, represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C \#define construct.

• The notation \[EABCD\] is used to identify an error value EABCD.

• Syntax, code examples and user input in interactive examples are shown in fixed width font. Brackets shown in this font, \[\], are part of the syntax and do not indicate optional items. In syntax the | symbol is used to separate alternatives, and ellipses (...) are used to show that additional arguments are optional.

• Bold fixed width font is used to identify brackets that surround optional items in syntax, \[\], and to identify system output in interactive examples.

• Variables within syntax statements are shown in italic fixed width font.

• Ranges of values are indicated with parentheses or brackets as follows:
  – (a,b) means the range of all values from a to b, including neither a nor b
  – [a,b] means the range of all values from a to b, including a and b
  – [a,b) means the range of all values from a to b, including a, but not b
  – (a,b] means the range of all values from a to b, including b, but not a.

Notes:
• Symbolic limits are used in this document instead of fixed values for portability. The values of most of these constants are defined in \<limits.h\> or \<unistd.h\>.
• The values of errors are defined in \<errno.h\>.

Other documents

The following documents are referenced in this specification:
• ANSI standard X3.159-1989, Programming Language C.
• System V Interface Definition (Spring 1986 - Issue 2).
• System V Release 2.0

Where to find more information

For an overview of the information associated with z/OS, see z/OS Information Roadmap.

Information updates on the web

For the latest information updates that have been provided in PTF cover letters and documentation APARs for z/OS®, see the online document at: http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/Shelves/ZDOCAPAR
This document is updated weekly and lists documentation changes before they are incorporated into z/OS publications.

**The z/OS Basic Skills Information Center**

The z/OS Basic Skills Information Center is a Web-based information resource intended to help users learn the basic concepts of z/OS, the operating system that runs most of the IBM mainframe computers in use today. The Information Center is designed to introduce a new generation of Information Technology professionals to basic concepts and help them prepare for a career as a z/OS professional, such as a z/OS system programmer.

Specifically, the z/OS Basic Skills Information Center is intended to achieve the following objectives:

- Provide basic education and information about z/OS without charge
- Shorten the time it takes for people to become productive on the mainframe
- Make it easier for new people to learn z/OS.

To access the z/OS Basic Skills Information Center, open your Web browser to the following Web site, which is available to all users (no login required): [http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp](http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp)
Summary of Changes

Summary of Changes for
SA38-0690-00
z/OS Version 2 Release 1

This book contains information also presented in OS/390 C Curses, SC28-1907-01.
Chapter 1. The Curses Library

The Curses library provides a set of functions that enable you to manipulate a terminal's display regardless of the terminal type. Throughout this documentation, the Curses library is referred to as curses. The basis of curses programming is the window data structure. Using this structure, you can manipulate data on a terminal's display. You can instruct curses to treat the entire terminal display as one large window or you can create multiple windows on the display. The windows can be different sizes and can overlap one another. The following figure shows a typical curses application with a single large window and one subwindow.

<table>
<thead>
<tr>
<th>TTY Type</th>
<th>Move cursor to desired item and press Enter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>tty rs232</td>
<td>Asynchronous Terminal</td>
</tr>
<tr>
<td>tty rs422</td>
<td>Asynchronous Terminal</td>
</tr>
<tr>
<td>F1=Help</td>
<td>F2=Refresh</td>
</tr>
<tr>
<td>F8=Image</td>
<td>F10=Exit</td>
</tr>
<tr>
<td>/=Find</td>
<td>N=Find Next</td>
</tr>
<tr>
<td>F9</td>
<td></td>
</tr>
</tbody>
</table>

Each window on a terminal's display has its own window data structure. This structure keeps state information about the window such as its size and where it is located on the display. Curses uses the window data structure to obtain relevant information it needs to carry out your instructions.

Terminology

When programming with curses, you should be familiar with the following terms:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>current character</td>
<td>The character that the logical cursor is currently on.</td>
</tr>
<tr>
<td>current line</td>
<td>The line that the logical cursor is currently on.</td>
</tr>
<tr>
<td>curscr</td>
<td>A virtual default window provided by curses. The curscr (current screen) is an internal representation of what currently appears on the terminal's external display. You should not modify the curscr.</td>
</tr>
</tbody>
</table>
display
A physical display connected to a workstation.

logical cursor
The cursor location within each window. The window data structure keeps track of the location of its logical cursor.

pad
A type of window that is larger than the dimensions of the terminal’s display. Unlike other windows, a pad is not associated with any particular portion of the display.

physical cursor
The cursor that appears on a display. The workstation uses this cursor to write to the display. There is only one physical cursor per display. To change the position of the physical cursor, you must do a refresh.

screen
The window that fills the entire display. The screen is synonymous with the stdscr (standard screen).

stdscr
A virtual default window provided by curses that represents the entire display.

window
A pointer to a C data structure and the graphic representation of that data structure on the display. A window can be thought of as a two-dimensional array representing how all or part of the display looks at any point in time. Windows range in size from the entire display to a single character.

**Naming Conventions**

A single curses function can have two or more versions. Curses functions with multiple versions follow distinct naming conventions that identify the separate versions. These conventions add a prefix to a standard curses function and identify what arguments the function requires or what actions take place when the function is called. The different versions of curses function names use three prefixes:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>Identifies a function that requires a window argument.</td>
</tr>
<tr>
<td>p</td>
<td>Identifies a function that requires a pad argument.</td>
</tr>
<tr>
<td>mv</td>
<td>Identifies a function that first performs a move to the program-supplied coordinates.</td>
</tr>
</tbody>
</table>

Some curses functions with multiple versions do not include one of the preceding prefixes. These functions use the curses default window stdscr (standard screen). The majority of functions that use the stdscr are functions created in the /usr/include/curses.h file using #define statements. The preprocessor replaces these statements at compilation time. As a result, these functions do not appear in the compiled assembly code, a trace, a debugger, or the curses source code.

If a curses function has only a single version, it does not necessarily use stdscr. For example, the `printw()` function prints a string to the stdscr. The `wprintw()` function prints a string to a specific window by supplying the Window argument. The `mvprintw()` function moves the specified coordinates to the stdscr and then performs the same function as the `printw()` function. Likewise, the `mvwprintw()` function moves the specified coordinates to the specified window and then performs the same function as the `wprintw()` function.
A function with the basic name is often provided for historical compatibility and operates only on single-byte characters. A function with the same name plus the \texttt{w} infix operates on wide (multi-byte) characters. A function with the same name plus the \texttt{_w} infix operates on complex characters and their renditions.

When a function with the same basic name operates on a single character, there is sometimes a function with the same name plus the \texttt{n} infix that operates on multiple characters. An \texttt{n} argument specifies the number of characters to process. The respective manual page specifies the outcome if the value of \texttt{n} is inappropriate.

\section*{Structure of a Curses Program}

In general, a curses program has the following progression:
- Start curses.
- Check for color support (optional).
- Start color (optional).
- Create one or more windows.
- Manipulate windows.
- Destroy one or more windows.
- Stop curses.

Your program does not have to follow this progression exactly.

\section*{Return Values}

With a few exceptions, all curses functions return either the integer value \texttt{ERR} or the integer value \texttt{OK}. Subroutines that do not follow this convention are noted appropriately. Subroutines that return pointers always return a null pointer on an error.
Chapter 2. Initializing Curses

You must include the `curses.h` file at the beginning of any program that calls curses functions. To do this, use the following statement:

```c
#include <curses.h>
```

Before you can call functions that manipulate windows or screens, you must call the `initscr()` or `newterm()` function. These functions first save the terminal’s settings. These functions then call the `setupterm()` function to establish a curses terminal.

Before exiting a curses program, you must call the `endwin()` function. The `endwin()` function restores tty modes, moves the cursor to the lower left corner of the screen, and resets the terminal into the proper nonvisual mode. You can also temporarily suspend curses. If you need to suspend curses, use a shell escape or system call for example. To resume after a temporary escape, you should call the `wrefresh()` or `doupdate()` function. The `isendwin()` function is helpful if, for optimization reasons, you don't want to call the `wrefresh()` function needlessly. You can determine if the `endwin()` function was called without any subsequent calls to the `wrefresh()` function by using the `isendwin()` function.

Most interactive, screen-oriented programs require character-at-a-time input without echoing the result to the screen. To establish your program with character-at-a-time input, call the `cbreak()` and `noecho()` functions after calling the `initscr` function. When accepting this type of input, programs should also call the following functions:

- `nonl()` function.
- `intrflush()` function with the Window parameter set to the stdscr and the Flag parameter set to `FALSE`. The Window parameter is required but ignored. You can use stdscr as the value of the Window parameter, because stdscr is already created for you.
- `keypad()` function with the Window parameter set to the stdscr and the Flag parameter set to `TRUE`. 
Chapter 3. Windows in the Curses Environment

A curses program manipulates windows that appear on a terminal’s display. A window is a rectangular portion of the display. A window can be as large as the entire display or as small as a single character in length and height.

Note: Pads are the exception. A pad is a window that is not restricted by the size of the screen. For more information, see “Pads” on page 9.

The following figure shows the different types of windows that exist in the curses environment:

![Diagram of different types of windows]

Within a curses program, windows are variables declared as type WINDOW. The WINDOW data type is defined in the `/usr/include/curses.h` file as a C data structure. You create a window by allocating a portion of a machine’s memory for a window structure. This structure describes the characteristics of the window. When a program changes the window data internally in memory, it must use the `wrefresh()` function (or equivalent function) to update the external, physical screen to reflect the internal change in the appropriate window structure.

Curses supplies a default window when the Curses library is initialized. You can create your own windows known as user-defined windows. Except for the amount of memory available to a program, there is no limit to the number of windows you can create. A curses program can manipulate the default window, user-defined windows, or both.

The Default Window Structure

Curses provides a virtual default window called stdscr. The stdscr represents, in memory, the entire terminal display. The stdscr window structure is created automatically when the Curses library is initialized and it describes the display. When the library is initialized, the length and width variables are set to the length and width of the physical display.

In addition to the stdscr, you can define your own windows. These windows are known as user-defined windows to distinguish them from the stdscr. Like the stdscr, user-defined windows exist in machine memory as structures.
Programs that use the stdscr first manipulate the stdscr and then call the `refresh()` function to refresh the external display so that it matches the stdscr window.

## The Current Window Structure

Curses also supports another virtual window called curscr (current screen). The curscr window is an internal representation of what currently appears on the terminal’s external display.

When a program requires the external representation to match the internal representation, it must call a function, such as the `wrefresh()` function, to update the physical display (or the `refresh()` function if the program is working with the stdscr). When a refresh is called on an internal window, curses copies the changed portions of the window into the curscr and updates the physical display.

The curscr is reserved for internal use by curses. You should not manipulate the curscr.

## Subwindows

Curses also allows you to construct subwindows. Subwindows are rectangular portions within other windows. A subwindow is also of type WINDOW. The window that contains a subwindow is known as the subwindow’s parent and the subwindow is known as the containing window’s child. The following figure demonstrates the parent child relationship.

Changes to either the parent window or the child window within the area overlapped by the subwindow are made to both windows. After modifying a subwindow, you should call the `touchline()` or `touchwin()` function on the parent window before refreshing it. The `touchline()` and `touchwin()` functions instruct curses to discard its optimization information for the parent window and to consider the window as having changed. A refresh called on the parent refreshes the children as well.

A subwindow can also be a parent window. The process of layering windows inside of windows is called nesting. The number of nested subwindows is limited to the amount of memory available up to the value of SHRT_MAX as defined in the `/usr/include/limits.h` file. Before you can delete a parent window, you must first delete all of its children using the `delwin()` function. Curses returns an error if you try to delete a window before removing all of its children.
Pads

A pad is a type of window that is not restricted by the terminal’s display size or associated with a particular part of the display. You can use pads whenever your program requires a large window. Because a pad is usually larger than the physical display, only a portion of a pad is visible to the user at a given time.

Use pads when you have a large amount of related data that you want to keep all together in one window but you do not need to display all of the data at once.

Windows within pads are known as subpads. Subpads are positioned within a pad at coordinates relative to the parent pad. This placement differs from subwindows which are positioned using screen coordinates.

You should use the `prefresh()` function to show a portion of a pad on the display. Unlike other windows, scrolling or echoing of input does not automatically refresh a pad. Like subwindows, when changing the image of a subpad, you must call either the `touchline()` or `touchwin()` function on the parent pad before refreshing the parent. You can use all the curses function with pads except for the `newwin()`, `subwin()`, `wrefresh()`, and `wnoutrefresh()` functions. These functions are replaced with the `newpad()`, `subpad()`, `prefresh()`, and `pnoutrefresh()` functions.
Chapter 4. Manipulating Window Data with Curses

When curses is initialized, the stdscr is provided automatically. You can manipulate the stdscr using the curses function library or you can create your own, user-defined windows.
Chapter 5. Curses Interfaces

This chapter describes the Curses functions, macros and external variables to support application portability at the C-language source level. The interface definitions are collated as though any underscore characters were not present.

addch()

Name
addch, mvaddch, mvwaddch, waddch - add a single-byte character and rendition to a window and advance the cursor

Synopsis
#include <curses.h>

int addch(const chtype ch);
int mvaddch(int y, int x, const chtype ch);
int mvwaddch(WINDOW *win, int y, int x, const chtype ch);
int waddch(WINDOW *win, const chtype ch);

Description
The addch(), mvaddch(), mvwaddch() and waddch() functions place ch into the current or specified window at the current or specified position, and then advance the window's cursor position. These functions perform wrapping. These functions perform special-character processing.

Return Value
Upon successful completion, these functions return OK. Otherwise they return ERR.

Errors
No errors are defined.

Application Usage
These functions are only guaranteed to operate reliably on character sets in which each character fits into a single byte, whose attributes can be expressed using only constants with the A_ prefix.

See Also
add_wch(), attroff(), doupdate(), <curses.h>.
addchstr()

**Name**
addchstr, addchnstr, mvaddchstr, mvaddchnstr, mvwaddchstr, mvwaddchnstr, waddchstr, waddchnstr - add string of single-byte characters and renditions to a window

**Synopsis**
```
#include <curses.h>

int addchstr(const chtype *chstr);
int addchnstr(const chtype *chstr, int n);
int mvaddchstr(int y, int x, const chtype *chstr);
int mvaddchnstr(int y, int x, const chtype *chstr, int n);
int mvwaddchstr(WINDOW *win, int y, int x, const chtype *chstr);
int mvwaddchnstr(WINDOW *win, int y, int x, const chtype *chstr, int n);
int waddchstr(WINDOW *win, const chtype *chstr);
int waddchnstr(WINDOW *win, const chtype *chstr, int n);
```

**Description**
These functions overlay the contents of the current or specified window, starting at the current or specified position, with the contents of the array pointed to by `chstr` until a null `cttype` is encountered in the array pointed to by `chstr`.

These functions do not change the cursor position. These functions do not perform special-character processing. These functions do not perform wrapping.

The `addchnstr()`, `mvaddchnstr()`, `mvwaddchnstr()` and `waddchnstr()` functions copy at most `n` items, but no more than will fit on the line. If `n` is -1 then the whole string is copied, to the maximum number that fit on the line.

**Return Value**
Upon successful completion, these functions return OK. Otherwise, they return ERR.

**Errors**
No errors are defined.

**Application Usage**
These functions are only guaranteed to operate reliably on character sets in which each character fits into a single byte, whose attributes can be expressed using only constants with the A_ prefix.

**See Also**
`addch()`, `add_wch()`, `add_wchstr()`, `<curses.h>`.
addnstr()

Name
addnstr, addstr, mvaddnstr, mvaddstr, mvwaddnstr, mvwaddstr waddnstr, waddstr
- add a string of multi-byte characters without rendition to a window and advance cursor

Synopsis
#include <curses.h>

int addnstr(const char *str, int n);
int addstr(const char *str);
int mvaddnstr(int y, int x, const char *str, int n);
int mvaddstr(int y, int x, const char *str);
int mvwaddnstr(WINDOW *win, int y, int x, char *const str, int n);
int mvwaddstr(WINDOW *win, int y, int x, char *const str);
int waddnstr(WINDOW *win, const char *str, int n);
int waddstr(WINDOW *win, const char *str);

Description
These functions write the characters of the string str on the current or specified window starting at the current or specified position using the background rendition.

These functions advance the cursor position. These functions perform special character processing. These functions perform wrapping.

The addstr(), mvaddstr(), mvwaddstr() and waddstr() functions are similar to calling mbstowcs() on str, and then calling addwstr(), mvaddwstr(), mvwaddwstr() and waddwstr(), respectively.

The addnstr(), mvaddnstr(), mvwaddnstr() and waddnstr() functions use at most n bytes from str. These functions add the entire string when n is -1. These functions are similar to calling mbstowcs() on the first n bytes of str, and then calling addwstr(), mvaddwstr(), mvwaddwstr() and waddwstr(), respectively.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

See Also
addnwstr(), mbstowcs(), <curses.h>
addnwstr()

Name

addnwstr, addwstr, mvaddnwstr, mvaddwstr, mvwaddnwstr, mvwaddwstr,
waddnwstr, waddwstr - add a wide-character string to a window and advance the
cursor

Synopsis

#include <curses.h>

int addnwstr(const wchar_t *wstr, int n);

int addwstr(const wchar_t *wstr);

int mvaddnwstr(int y, int x, const wchar_t *wstr, int n);

int mvaddwstr(int y, int x, const wchar_t *wstr);

int mvwaddnwstr(WINDOW *win, int y, int x, const wchar_t *wstr, int n);

int mvwaddwstr(WINDOW *win, int y, int x, const wchar_t *wstr);

int waddnwstr(WINDOW *win, const wchar_t *wstr, int n);

int waddwstr(WINDOW *win, const wchar_t *wstr);

Description

These functions write the characters of the wide character string wstr on the
current or specified window at that window’s current or specified cursor position.

These functions advance the cursor position. These functions perform special
character processing. These functions perform wrapping.

The effect is similar to building a cchar_t from the wchar_t and the background
rendition and calling wadd_wch(), once for each wchar_t character in the string.
The cursor movement specified by the mv functions occurs only once at the start
of the operation.

The addnwstr(), mvaddnwstr(), mvwaddnwstr() and waddnwstr() functions write
at most n wide characters. If n is -1, then the entire string will be added.

Return Value

Upon successful completion, these functions return OK. Otherwise, they return
ERR.

Errors

No errors are defined.

See Also

add_wch(), <curses.h>
add_wch()

Name
add_wch, mvadd_wch, mvwadd_wch, wadd_wch - add a complex character and
rendition to a window

Synopsis
#include <curses.h>
int add_wch(cchar_t *const wch);
int wadd_wch(WINDOW *win, cchar_t *const wch);
int mvadd_wch(int y, int x, cchar_t *const wch);
int mvwadd_wch(WINDOW *win, int y, int x, cchar_t *const wch);

Description
These functions add information to the current or specified window at the current
or specified position, and then advance the cursor. These functions perform
wrapping. These functions perform special-character processing.
• If wch refers to a spacing character, then any previous character at that location
  is removed, a new character specified by wch is placed at that location with
  rendition specified by wch; then the cursor advances to the next spacing
  character on the screen.
• If wch refers to a non-spacing character, all previous characters at that location
  are preserved, the non-spacing characters of wch are added to the spacing
  complex character, and the rendition specified by wch is ignored.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return
ERR.

Errors
No errors are defined.

See Also
addch(), <curses.h>.

add_wchnstr()

Name
add_wchnstr, add_wchstr, mvadd_wchnstr, mvwadd_wchstr, mvwadd_wchnstr,
mvadd_wchstr, wadd_wchnstr, wadd_wchstr - add an array of complex
characters and renditions to a window

Synopsis
#include <curses.h>

int add_wchnstr(const cchar_t *wchstr, int n);
int add_wchstr(const cchar_t *wchstr);
int wadd_wchnstr(WINDOW *win, const cchar_t *wchstr, int n);
int wadd_wchstr(WINDOW *win, const cchar_t *wchstr);
int mvadd_wchnstr(int y, int x, const cchar_t *wchstr, int n);
int mvadd_wchstr(int y, int x, const cchar_t *wchstr);
int mvwadd_wchnstr(WINDOW *win, int y, int x, const cchar_t *wchstr, int n);
int mvwadd_wchstr(WINDOW *win, int y, int x, const cchar_t *wchstr);

Description
These functions write the array of cchar_t specified by wchstr into the current or
specified window starting at the current or specified cursor position.

These functions do not advance the cursor. The results are unspecified if wchstr
contains any special characters.

The functions end successfully on encountering a null cchar_t. The functions also
end successfully when they fill the current line. If a character cannot completely fit
at the end of the current line, those columns are filled with the background
character and rendition.

The add_wchnstr(), mvadd_wchnstr(), mvwadd_wchnstr() and wadd_wchnstr()
fundations end successfully after writing n cchar_ts (or the entire array of cchar_ts,
if n is -1).

Return Value
Upon successful completion, these functions return OK. Otherwise, they return
ERR.

Errors
No errors are defined.

See Also
<curves.h>.

attroff()

Name
attroff, attron, attrset, wattroff, wattion, watrrset - restricted window attribute
control functions

Synopsis
#include <curves.h>
int attroff(int attrs);
int attron(int attrs);
int attrset(int attrs);
int wattroff(WINDOW *win, int attrs);
int wattron(WINDOW *win, int attrs);
int wattrset(WINDOW *win, int attrs);

Description

These functions manipulate the window attributes of the current or specified window.

The attroff() and wattroff() functions turn off attrs in the current or specified window without affecting any others.

The attron() and wattron() functions turn on attrs in the current or specified window without affecting any others.

The attrset() and wattrset() functions set the background attributes of the current or specified window to attrs.

It is unspecified whether these functions can be used to manipulate attributes other than A_BLINK, A_BOLD, A_DIM, A_REVERSE, A_STANDOUT and A_UNDERLINE.

Return Value

These functions always return either OK or 1.

Errors

No errors are defined.

See Also

attr_get(), standend(), <curses.h>.

attr_get()

Name

attr_get, attr_off, attr_on, attr_set, color_set, wattr_get, wattr_off, wattr_on, wattr_set, wcolor_set -- window attribute control functions

Synopsis

#include <curses.h>

int attr_get(attr_t *attrs, short *color_pair_number, void *opts);
int attr_off(attr_t attrs, void *opts);
int attr_on(attr_t attrs, void *opts);
int attr_set(attr_t attrs, short color_pair_number, void *opts);
int color_set(short color_pair_number, void *opts);
in wattr_get (WINDOW *win, attr_t *attrs, short *color_pair_number, void *opts);
in wattr_off(WINDOW *win, attr_t attrs, void *opts);
in wattr_on(WINDOW *win, attr_t attrs, void *opts);
Enhanced Curses

```c
int wattr_set(WINDOW *win, attr_t attrs, short color_pair_number, void *opts);
int wcolor_set(WINDOW *win, short color_pair_number, void *opts);
```

**Description**

These functions manipulate the attributes and color of the window rendition of the current or specified window.

The `attr_get()` and `wattr_get()` functions obtain the current rendition of a window. If `attrs` or `color_pair_number` is a null pointer, no information will be obtained on the corresponding rendition information and this is not an error.

The `attr_off()` and `wattr_off()` functions turn off `attrs` in the current or specified window without affecting any others.

The `attr_on()` and `wattr_on()` functions turn on `attrs` in the current or specified window without affecting any others.

The `attr_set()` and `wattr_set()` functions set the window rendition of the current or specified window to `attrs` and `color_pair_number`.

The `color_set()` and `wcolor_set` functions set the window color of the current or specified window to `color_pair_number`.

**Return Value**

The `attr_get()` and `wattr_get()` functions return the current window attributes for the current or specified window.

The other functions always return OK.

**Errors**

No errors are defined.

**See Also**

`attroff()`, `<curses.h>`.

---

**baudrate()**

**Name**

`baudrate` - get terminal baud rate

**Synopsis**

```c
#include <curses.h>

int baudrate(void);
```

**Description**

The `baudrate()` function extracts the output speed of the terminal in bits per second.
Return Value
The baudrate() function returns the output speed of the terminal.

Errors
No errors are defined.

See Also
tcgetattr(), <curses.h>.

beep()

Name
beep - audible signal

Synopsis
#include <curses.h>

int beep(void);

Description
The beep() function alerts the user. It sounds the audible alarm on the terminal, or if that is not possible, it flashes the screen (visible bell). If neither signal is possible, nothing happens.

Return Value
The beep() function always returns OK.

Errors
No errors are defined.

Application Usage
Nearly all terminals have an audible alarm, but only some can flash the screen.

See Also
flash(), <curses.h>.

bkgd()

Name
bkgd, bkgdset, getbkgd, wbkgd, wbkgdset - turn off the previous background attributes, OR the requested attributes into the window rendition, and set or get background character and rendition using a single-byte character.

Synopsis
#include <curses.h>

int bkgd(chtype ch);

void bkgdset(chtype ch);
chtype getbkgd(WINDOW *win);
int wbkgd(WINDOW *win, chtype ch);
void wbkgdset(WINDOW *win, chtype ch);

Description
The bkgdset() and wbkgdset() functions turn off the previous background attributes, OR the requested attributes into the window rendition, and set the background attributes of the current or specified window based on the information in \textit{ch}. If \textit{ch} refers to a multi-column character, the results are undefined.

The bkgd() and wbkgd() functions turn off the previous background attributes, OR the requested attributes into the window rendition, and set the background property of the current or specified window and then apply this setting to every character position in that window:

\begin{itemize}
  \item The rendition of every character on the screen is changed to the new background rendition.
  \item Wherever the former background character appears, it is changed to the new background character.
\end{itemize}

The getbkgd() function extracts the specified window's background character and rendition.

Return Value
Upon successful completion, bkgd() and wbkgd() return OK. Otherwise, they return ERR.

The bkgdset() and wbkgdset() functions do not return a value.

Upon successful completion, getbkgd() returns the specified window's background character and rendition. Otherwise, it returns (chtype) ERR.

bkgd()

Errors
No errors are defined.

Application Usage
These functions are only guaranteed to operate reliably on character sets in which each character fits into a single byte, whose attributes can be expressed using only constants with the \texttt{A_} prefix.

See Also
<\texttt{curses.h}>.
**bkgrnd()**

### Name

`bkgrnd`, `bkgrndset`, `getbkgrnd`, `wbkgrnd`, `wbkgrndset`, `wgetbkgrnd` — turn off the previous background attributes, OR the requested attributes into the window rendition, and set or get background character and rendition using a complex character.

### Synopsis

```c
#include <curses.h>

int bkgrnd(const cchar_t *wch);
void bkgrndset(const cchar_t *wch);
int getbkgrnd(cchar_t *wch);
int wbkgrnd(WINDOW *win, const cchar_t *wch);
void wbkgrndset(WINDOW *win, const cchar_t *wch);
int wgetbkgrnd(WINDOW *win, cchar_t *wch);
```

### Description

The `bkgrndset()` and `wbkgrndset()` functions turn off the previous background attributes, OR the requested attributes into the window rendition, and set the background property of the current or specified window based on the information in `wch`.

The `bkgrnd()` and `wbkgrnd()` functions turn off the previous background attributes, OR the requested attributes into the window rendition, and set the background property of the current or specified window and then apply this setting to every character position in that window:

- The rendition of every character on the screen is changed to the new background rendition.
- Wherever the former background character appears, it is changed to the new background character.

If `wch` refers to a non-spacing complex character for `bkgrnd()`, `bkgrndset()`, `wbkgrnd()` and `wbkgrndset()`, then `wch` is added to the existing spacing complex character that is the background character. If `wch` refers to a multi-column character, the results are unspecified.

The `getbkgrnd()` and `wgetbkgrnd()` functions store, into the area pointed to by `wch`, the value of the window’s background character and rendition.

### Return Value

The `bkgrndset()` and `wbkgrndset()` functions do not return a value.

Upon successful completion, the other functions return `OK`. Otherwise, they return `ERR`.

### Errors

No errors are defined.
See Also

`<curses.h>`.

border()

Name

border, wborder - draw borders from single-byte characters and renditions

Synopsis

```
#include <curses.h>

int border(chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, 
chtype tr, chtype bl, chtype br);

int wborder(WINDOW *win, chtype ls, chtype rs, chtype ts, chtype bs, 
chtetype tl, chtype tr, chtype bl, chtype br);
```

Description

The `border()` and `wborder()` functions draw a border around the edges of the current or specified window. These functions do not advance the cursor position. These functions do not perform special character processing. These functions do not perform wrapping.

The arguments in the left-hand column of the following table contain single-byte characters with renditions, which have the following uses in drawing the border:

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Usage</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ls</code></td>
<td>Starting-column side</td>
<td>ACS_VLINE</td>
</tr>
<tr>
<td><code>rs</code></td>
<td>Ending-column side</td>
<td>ACS_VLINE</td>
</tr>
<tr>
<td><code>ts</code></td>
<td>First-line side</td>
<td>ACS_HLINE</td>
</tr>
<tr>
<td><code>bs</code></td>
<td>Last-line side</td>
<td>ACS_HLINE</td>
</tr>
<tr>
<td><code>tl</code></td>
<td>Corner of the first line and the starting column</td>
<td>ACS_ULCORNER</td>
</tr>
<tr>
<td><code>tr</code></td>
<td>Corner of the first line and the ending column</td>
<td>ACS_URCORNER</td>
</tr>
<tr>
<td><code>bl</code></td>
<td>Corner of the last line and the starting column</td>
<td>ACS_BLCORNER</td>
</tr>
<tr>
<td><code>br</code></td>
<td>Corner of the last line and the ending column</td>
<td>ACS_BRCORNER</td>
</tr>
</tbody>
</table>

If the value of any argument in the left-hand column is 0, then the default value in the right-hand column is used. If the value of any argument in the left-hand column is a multi-column character, the results are undefined.

Return Value

Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors

No errors are defined.
**Application Usage**

These functions are only guaranteed to operate reliably on character sets in which each character fits into a single byte, whose attributes can be expressed using only constants with the A_ prefix.

**See Also**

border_set(), box(), hline(), <curses.h>.

---

### border_set()

**Name**

border_set, wborder_set, - draw borders from complex characters and renditions

**Synopsis**

```c
#include <curses.h>

int border_set(const cchar_t *ls, const cchar_t *rs, const cchar_t *ts, 
               const cchar_t *bs, const cchar_t *tl, const cchar_t *tr, 
               const cchar_t *bl, const cchar_t *br);

int wborder_set(WINDOW *win, const cchar_t *ls, const cchar_t *rs, 
                const cchar_t *ts, const cchar_t *bs, 
                const cchar_t *tl, const cchar_t *tr, 
                const cchar_t *bl, const cchar_t *br);
```

**Description**

The border_set() and wborder_set() functions draw a border around the edges of the current or specified window. These functions do not advance the cursor position. These functions do not perform special character processing. These functions do not perform wrapping.

The arguments in the left-hand column of the following table contain spacing complex characters with renditions, which have the following uses in drawing the border:

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Usage</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ls</td>
<td>Starting-column side</td>
<td>WACS_VLINE</td>
</tr>
<tr>
<td>rs</td>
<td>Ending-column side</td>
<td>WACS_VLINE</td>
</tr>
<tr>
<td>ts</td>
<td>First-line side</td>
<td>WACS_HLINE</td>
</tr>
<tr>
<td>bs</td>
<td>Last-line side</td>
<td>WACS_HLINE</td>
</tr>
<tr>
<td>tl</td>
<td>Corner of the first line and the starting column</td>
<td>WACS_ULCORNER</td>
</tr>
<tr>
<td>tr</td>
<td>Corner of the first line and the ending column</td>
<td>WACS_URCORNER</td>
</tr>
<tr>
<td>bl</td>
<td>Corner of the last line and the starting column</td>
<td>WACS_BLCORNER</td>
</tr>
<tr>
<td>br</td>
<td>Corner of the last line and the ending column</td>
<td>WACS_BRCORNER</td>
</tr>
</tbody>
</table>
If the value of any argument in the left-hand column is a null pointer, then the default value in the right-hand column is used. If the value of any argument in the left-hand column is a multi-column character, the results are undefined.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

See Also
box_set(), hline_set(), <curses.h>.

box()

Name
box - draw borders from single-byte characters and renditions

Synopsis
#include <curses.h>

int box(WINDOW *win, chtype verch, chtype horch);

Description
The box() function draws a border around the edges of the specified window. This function does not advance the cursor position. This function does not perform special character processing. This function does not perform wrapping.

The function box(win, verch, horch) has an effect equivalent to:
wborder(win, verch, verch, horch, horch, 0, 0, 0, 0);

Return Value
Upon successful completion, box() returns OK. Otherwise, it returns ERR.

Errors
No errors are defined.

Application Usage
These functions are only guaranteed to operate reliably on character sets in which each character fits into a single byte, whose attributes can be expressed using only constants with the A_ prefix.

See Also
border(), box_set(), hline(), <curses.h>.
box_set()

Name
box_set - draw borders from complex characters and renditions

Synopsis
#include <curses.h>

int box_set(WINDOW *win, const cchar_t *verch, const cchar_t *horch);

Description
The box_set() function draws a border around the edges of the specified window. This function does not advance the cursor position. This function does not perform special character processing. This function does not perform wrapping.

The function box_set(win, verch, horch) has an effect equivalent to:

wborder_set(win, verch, verch, horch, horch, NULL, NULL, NULL, NULL);

Return Value
Upon successful completion, this function returns OK. Otherwise, it returns ERR.

Errors
No errors are defined.

See Also
border_set(), hline_set(), <curses.h>.

can_change_color()

Name
can_change_color, color_content, has_colors, init_color, init_pair, start_color, pair_content — color manipulation functions

Synopsis
#include <curses.h>

bool can_change_color(void);

int color_content(short color, short *red, short *green, short *blue);

int COLOR_PAIR(int n);

bool has_colors(void);

int init_color(short color, short red, short green, short blue);

int init_pair(short pair, short f, short b);

int pair_content(short pair, short *f, short *b);

int PAIR_NUMBER(int value);

int start_color(void);
extern int COLOR_PAIRS;
extern int COLORS;

Description
These functions manipulate color on terminals that support color.

Querying Capabilities
The has_colors() function indicates whether the terminal is a color terminal. The can_change_color() function indicates whether the terminal is a color terminal on which colors can be redefined.

Initialization
The start_color() function must be called in order to enable use of colors and before any color manipulation function is called. The function initializes eight basic colors (black, blue, green, cyan, red, magenta, yellow, and white) that can be specified by the color macros (such as COLOR_BLACK) defined in <curses.h>. The initial appearance of these eight colors is not specified.

The function also initializes two global external variables:
- COLORS defines the number of colors that the terminal supports. (See Color Identification below.) If COLORS is 0, the terminal does not support redefinition of colors (and can_change_color() will return FALSE).
- COLOR_PAIRS defines the maximum number of color-pairs that the terminal supports. (See User-Defined Color Pairs below.)

The start_color() function also restores the colors on the terminal to terminal-specific initial values. The initial background color is assumed to be black for all terminals.

Color Identification
The init_color() function redefines color number color, on terminals that support the redefinition of colors, to have the red, green, and blue intensity components specified by red, green, and blue, respectively. Calling init_color() also changes all occurrences of the specified color on the screen to the new definition.

The color_content() function identifies the intensity components of color number color. It stores the red, green, and blue intensity components of this color in the addresses pointed to by red, green, and blue, respectively.

For both functions, the color argument must be in the range from 0 to and including COLORS-1. Valid intensity values range from 0 (no intensity component) up to and including 1000 (maximum intensity in that component).

User-Defined Color Pairs
Calling init_pair() defines or redefines color-pair number pair to have foreground color f and background color b. Calling init_pair() changes any characters that were displayed in the color pair’s old definition to the new definition and refreshes the screen.

After defining the color pair, the macro COLOR_PAIR(n) returns the value of color pair n. This value is the color attribute as it would be extracted from a chtype. Conversely, the macro PAIR_NUMBER(value) returns the color pair number associated with the color attribute value.
The pair_content() function retrieves the component colors of a color-pair number pair. It stores the foreground and background color numbers in the variables pointed to by \( f \) and \( b \), respectively.

With init_pair() and pair_content(), the value of pair must be in a range from 0 to and including COLOR_PAIRS-1. (There may be an implementation-specific lower limit on the valid value of pair, but any such limit is at least 63.) Valid values for \( f \) and \( b \) are the range from 0 to and including COLORS-1.

### Return Value

The has_colors() function returns TRUE if the terminal can manipulate colors; otherwise, it returns FALSE.

The can_change_color() function returns TRUE if the terminal supports colors and can change their definitions; otherwise, it returns FALSE.

Upon successful completion, the other functions return OK; otherwise, they return ERR.

### Errors

No errors are defined.

### Application Usage

To use these functions, start_color() must be called, usually right after initscr().

The can_change_color() and has_colors() functions facilitate writing terminal-independent programs. For example, a programmer can use them to decide whether to use color or some other video attribute.

On color terminals, a typical value of COLORS is 8 and the macros such as COLOR_BLACK return a value within the range from 0 to and including 7. However, applications cannot rely on this to be true.

### See Also

attroff(), delscreen(), <curses.h>.

cbreak()

### Name

cbreak, nocbreak, noraw, raw - input mode control functions

### Synopsis

```c
#include <curses.h>
int cbreak(void);
int nocbreak(void);
int noraw(void);
int raw(void);
```
Curses

Description
The cbreak() function sets the input mode for the current terminal to cbreak mode and overrides a call to raw().

The nocbreak() function sets the input mode for the current terminal to Cooked Mode without changing the state of ISIG and IXON.

The noraw() function sets the input mode for the current terminal to Cooked Mode and sets the ISIG and IXON flags.

The raw() function sets the input mode for the current terminal to Raw Mode.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

Application Usage
If the application is not certain what the input mode of the process was at the time it called initscr(), it should use these functions to specify the desired input mode.

See Also
<curtse.h>.

chgat()

Name
chgat, mvchgt, mvwchgat, wchgt - change renditions of characters in a window

Synopsis
#include <curses.h>

int chgat(int n, attr_t attr, short color, const void *opts);
int mvchgt(int y, int x, int n, attr_t attr, short color,
            const void *opts);
int mvwchgt(WINDOW *win, int y, int x, int n, attr_t attr,
           short color, const void *opts);
int wchgt(WINDOW *win, int n, attr_t attr, short color,
          const void *opts);

Description
These functions change the renditions of the next n characters in the current or specified window (or of the remaining characters on the line, if n is -1), starting at the current or specified cursor position. The attributes and colors are specified by attr and color as for setcchar().

These functions do not update the cursor. These functions do not perform wrapping.
A value of $n$ that is greater than the remaining characters on a line is not an error.

The `opts` argument is reserved for definition in a future edition of this document. Currently, the application must provide a null pointer as `opts`.

**Return Value**

Upon successful completion, these functions return OK. Otherwise, they return ERR.

**Errors**

No errors are defined.

**See Also**

`setcchar()`, `<curses.h>`

---

**clear()**

**Name**

clear, erase, wclear, werase - clear a window

**Synopsis**

```c
#include <curses.h>
int clear(void);
int erase(void);
int wclear(WINDOW *win);
int werase(WINDOW *win);
```

**Description**

The clear(), erase(), wclear() and werase() functions clear every position in the current or specified window.

The clear() and wclear() functions also achieve the same effect as calling clearok(), so that the window is cleared completely on the next call to wrefresh() for the window and is redrawn in its entirety.

**Return Value**

Upon successful completion, these functions return OK. Otherwise, they return ERR.

**Errors**

No errors are defined.

**See Also**

`clearok()`, `doupdate()`, `<curses.h>`.
clearok()

**Name**
clearok, idlok, leaveok, scrollok, setscrreg, wsetscrreg - terminal output control functions

**Synopsis**
```
#include <curses.h>

int clearok(WINDOW *win, bool bf);
int idlok(WINDOW *win, bool bf);
int leaveok(WINDOW *win, bool bf);
int scrollok(WINDOW *win, bool bf);
int setscrreg(int top, int bot);
int wsetscrreg(WINDOW *win, int top, int bot);
```

**Description**
These functions set options that deal with output within Curses.

The clearok() function assigns the value of \textit{bf} to an internal flag in the specified window that governs clearing of the screen during a refresh. If, during a refresh operation on the specified window, the flag in curscr is TRUE or the flag in the specified window is TRUE, then the implementation clears the screen, redraws it in its entirety, and sets the flag to FALSE in curscr and in the specified window. The initial state is unspecified.

The idlok() function specifies whether the implementation may use the hardware insert-line, delete-line, and scroll features of terminals so equipped. If \textit{bf} is TRUE, use of these features is enabled. If \textit{bf} is FALSE, use of these features is disabled and lines are instead redrawn as required. The initial state is FALSE.

The leaveok() function controls the cursor position after a refresh operation. If \textit{bf} is TRUE, refresh operations on the specified window may leave the terminal's cursor at an arbitrary position. If \textit{bf} is FALSE, then at the end of any refresh operation, the terminal's cursor is positioned at the cursor position contained in the specified window. The initial state is FALSE.

The scrollok() function controls the use of scrolling. If \textit{bf} is TRUE, then scrolling is enabled for the specified window. If \textit{bf} is FALSE, scrolling is disabled for the specified window. The initial state is FALSE.

The setscrreg() and wsetscrreg() functions define a software scrolling region in the current or specified window. The \textit{top} and \textit{bot} arguments are the line numbers of the first and last line defining the scrolling region. (Line 0 is the top line of the window.) If this option and scrollok() are enabled, an attempt to move off the last line of the margin causes all lines in the scrolling region to scroll one line in the direction of the first line. Only characters in the window are scrolled. If a software scrolling region is set and scrollok() is not enabled, an attempt to move off the last line of the margin does not reposition any lines in the scrolling region.
Return Value

Upon successful completion, setscrreg() and wsetscrreg() return OK. Otherwise, they return ERR.

The other functions always return OK.

Errors

No errors are defined.

Application Usage

The only reason to enable the idlok() feature is to use scrolling to achieve the visual effect of motion of a partial window, such as for a screen editor. In other cases, the feature can be visually annoying.

The leaveok() option provides greater efficiency for applications that do not use the cursor.

See Also

clear(), delscreen(), doupdate(), scrl(), <curses.h>

clrtobot()
clrtoeol()

Name
clrtoeol, wclrtoeol - clear from cursor to end of line

Synopsis
#include <curses.h>
int clrtoeol(void);
int wclrtoeol(WINDOW *win);

Description
The clrtoeol() and wclrtoeol() functions erase the current line from the cursor to the end of the line, inclusive, in the current or specified window.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

See Also
doupdate(), <curses.h>.

color_content()

Name
color_content - identify red/green/blue intensity of a color

Synopsis
#include <curses.h>
int color_content(short color, short *red, short *green, short *blue);

Description
Refer to can_change_color()

COLOR_PAIRS

Name
COLOR_PAIRS, COLORS - external variables for color support

Synopsis
#include <curses.h>
extern int COLOR_PAIRS;
extern int COLORS;
Description
Refer to `can_change_color()`.

**COLS**

**Name**

COLS - number of columns on terminal screen

**Synopsis**

```
#include <curses.h>

extern int COLS;
```

**Description**

The external variable `COLS` indicates the number of columns on the terminal screen.

**See Also**

`initscr()`, `<curses.h>`.

**copywin()**

**Name**

copywin - copy a region of a window

**Synopsis**

```
#include <curses.h>

int copywin(const WINDOW *srcwin, WINDOW *dstwin, int sminrow, int smincol, int dminrow, int dmincol, int dmaxrow, int dmaxcol, int overlay);
```

**Description**

The `copywin()` function provides a finer granularity of control over the `overlay()` and `overwrite()` functions. As in the `prefresh()` function, a rectangle is specified in the destination window, `(dminrow, dmincol)` and `(dmaxrow, dmaxcol)`, and the upper-left-corner coordinates of the source window, `(sminrow, smincol)`. If `overlay` is TRUE, then copying is non-destructive, as in `overlay()`. If `overlay` is FALSE, then copying is destructive, as in `overwrite()`.

**Return Value**

Upon successful completion, `copywin()` returns OK. Otherwise, it returns ERR.

**Errors**

No errors are defined.

**See Also**

`newpad()`, `overlay()`, `<curses.h>`.
curscr

Name
curscr - current window

Synopsis
#include <curses.h>
extern WINDOW *curscr;

Description
The external variable curscr points to an internal data structure. It can be specified
as an argument to certain functions, such as clearok(), where permitted in this
specification.

See Also
clearok(), <curses.h>.

curs_set()

Name
curs_set - set the cursor mode

Synopsis
#include <curses.h>

int curs_set(int visibility);

Description
The curs_set() function sets the appearance of the cursor based on the value of
visibility:

<table>
<thead>
<tr>
<th>Value of visibility</th>
<th>Appearance of Cursor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Invisible</td>
</tr>
<tr>
<td>1</td>
<td>Terminal-specific normal mode</td>
</tr>
<tr>
<td>2</td>
<td>Terminal-specific high visibility mode</td>
</tr>
</tbody>
</table>

The terminal does not necessarily support all the above values.

Return Value
If the terminal supports the cursor mode specified by visibility, then curs_set()
returns the previous cursor state. Otherwise, the function returns ERR.

Errors
No errors are defined.

See Also
<curses.h>.
cur_term()

Name
cur_term - current terminal information

Synopsis
#include <term.h>
extern TERMINAL *cur_term;

Description
The external variable cur_term identifies the record in the terminfo database associated with the terminal currently in use.

See Also
set_curterm(), tigetflag(), <term.h>.

def_prog_mode()

Name
def_prog_mode, def_shell_mode, reset_prog_mode, reset_shell_mode - save/restore program or shell terminal modes

Synopsis
#include <curses.h>
int def_prog_mode(void);
int def_shell_mode(void);
int reset_prog_mode(void);
int reset_shell_mode(void);

Description
The def_prog_mode() function saves the current terminal modes as the “program” (in Curses) state for use by reset_prog_mode().

The def_shell_mode() function saves the current terminal modes as the “shell” (not in Curses) state for use by reset_shell_mode().

The reset_prog_mode() function restores the terminal to the “program” (in Curses) state.

The reset_shell_mode() function restores the terminal to the “shell” (not in Curses) state.

These functions affect the mode of the terminal associated with the current screen.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.
Errors
No errors are defined.

Application Usage
The initscr() function achieves the effect of calling def_shell_mode() to save the prior terminal settings so they can be restored during the call to endwin(), and of calling def_prog_mode() to specify an initial definition of the program terminal mode.

Applications normally do not need to refer to the shell terminal mode. Applications may find it useful to save and restore the program terminal mode.

See Also
doupdate(), endwin(), initscr(), <curses.h>.

delay_output()

Name
delay_output - delay output

Synopsis
#include <curses.h>
int delay_output(int ms);

Description
On terminals that support pad characters, delay_output() pauses the output for at least ms milliseconds. Otherwise, the length of the delay is unspecified.

Return Value
Upon successful completion, delay_output() returns OK. Otherwise, it returns ERR.

Errors
No errors are defined.

Application Usage
Whether or not the terminal supports pad characters, the delay_output() function is not a precise method of timekeeping.

See Also
napms(), <curses.h>.

delch()

Name
delch, mvdelch, mvwdelch, wdelch - delete a character from a window.
Synopsis
#include <curses.h>

int delch(void);
int mvdelch(int y, int x);
int mvwdelch(WINDOW *win, int y, int x);
int wdelch(WINDOW *win);

Description
These functions delete the character at the current or specified position in the current or specified window. This function does not change the cursor position.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

See Also
#include <curses.h>.

del_curterm()

Name
del_curterm, restartterm, set_curterm, setupterm - interfaces to the terminfo database

Synopsis
#include <term.h>

int del_curterm(TERMINAL *oterm);
int restartterm(char *term, int fildes, int *errret);
TERMINAL *set_curterm(TERMINAL *nterm);
int setupterm(char *term, int fildes, int *errret);
extern TERMINAL *cur_term;

Description
These functions retrieve information from the terminfo database.

To gain access to the terminfo database, setupterm() must be called first. It is automatically called by initscr() and newterm(). The setupterm() function initializes the other functions to use the terminfo record for a specified terminal (which depends on whether use_env() was called). It sets the cur_term external variable to a TERMINAL structure that contains the record from the terminfo database for the specified terminal.
The terminal type is the character string term; if term is a null pointer, the environment variable TERM is used. If TERM is not set or if its value is an empty string, then "unknown" is used as the terminal type. The application must set fildes to a file descriptor, open for output, to the terminal device, before calling setupterm(). If errret is not null, the integer it points to is set to one of the following values to report the function outcome:

-1  The terminfo database was not found (function fails).
0   The entry for the terminal was not found in terminfo (function fails).
1   Success.

If setupterm() detects an error and errret is a null pointer, setupterm() writes a diagnostic message and exits.

A simple call to setupterm() that uses all the defaults and sends the output to stdout is:
setupterm((char *)0, fileno(stdout), (int *)0);

The set_curterm() function sets the variable cur_term to nterm, and makes all of the terminfo boolean, numeric, and string variables use the values from nterm.

The del_curterm() function frees the space pointed to by oterm and makes it available for further use. If oterm is the same as cur_term, references to any of the terminfo boolean, numeric, and string variables thereafter may refer to invalid memory locations until setupterm() is called again.

The restartterm() function assumes a previous call to setupterm() (perhaps from initscr() or newterm()). It lets the application specify a different terminal type in term and updates the information returned by baudrate() based on fildes, but does not destroy other information created by initscr(), newterm() or setupterm().

**Return Value**

Upon successful completion, set_curterm() returns the previous value of cur_term. Otherwise, it returns a null pointer.

Upon successful completion, the other functions return OK. Otherwise, they return ERR.

**Errors**

No errors are defined.

**Application Usage**

An application would call setupterm() if it required access to the terminfo database but did not otherwise need to use Curses.

**See Also**

baudrate(), erasechar(), has_ic(), longname(), putc(), termattrs(), termmname(), tgetent(), tigetflag(), use_env(), <term.h>.
deleteIn()

Name
deleteIn, wdeleteIn - delete lines in a window

Synopsis
#include <curses.h>
int deleteIn(void);
int wdeleteIn(WINDOW *win);

Description
The deleteIn() and wdeleteIn() functions delete the line containing the cursor in the
current or specified window and move all lines following the current line one line
toward the cursor. The last line of the window is cleared. The cursor position does
not change.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return
ERR.

Errors
No errors are defined.

See Also
insdellIn(), <curses.h>.

delscreen()

Name
delscreen - free storage associated with a screen

Synopsis
#include <curses.h>
void delscreen(SCREEN *sp);

Description
The delscreen() function frees storage associated with the SCREEN pointed to by
sp.

Return Value
The delscreen() function does not return a value.

Errors
No errors are defined.

See Also
endwin(), initscr(), <curses.h>.
delwin()

Name
delwin - delete a window

Synopsis
#include <curses.h>
int delwin(WINDOW *win);

Description
The delwin() function deletes \textit{win}, freeing all memory associated with it. The application must delete subwindows before deleting the main window.

Return Value
Upon successful completion, delwin() returns OK. Otherwise, it returns ERR.

Errors
No errors are defined.

See Also
\textit{derwin()}, \textit{dupwin()}, \textit{<curses.h>}

derwin()

Name
derwin, newwin, subwin - window creation functions

Synopsis
#include <curses.h>

WINDOW *derwin(WINDOW *orig, int nlines, int ncols, int begin_y,
               int begin_x);
WINDOW *newwin(int nlines, int ncols, int begin_y, int begin_x);
WINDOW *subwin(WINDOW *orig, int nlines, int ncols, int begin_y,
               int begin_x);

Description
The derwin() function is the same as subwin(), except that \textit{begin\_y} and \textit{begin\_x} are relative to the origin of the window \textit{orig} rather than absolute screen positions.

The newwin() function creates a new window with \textit{nlines} lines and \textit{ncols} columns, positioned so that the origin is \textit{(begin\_y, begin\_x)}. If \textit{nlines} is zero, it defaults to \textit{LINES - begin\_y}; if \textit{ncols} is zero, it defaults to \textit{COLS - begin\_x}.

The subwin() function creates a new window with \textit{nlines} lines and \textit{ncols} columns, positioned so that the origin is at \textit{(begin\_y, begin\_x)}. (This position is an absolute screen position, not a position relative to the window \textit{orig}.) If any part of the new window is outside \textit{orig}, the function fails and the window is not created.
Return Value
Upon successful completion, these functions return a pointer to the new window. Otherwise, they return a null pointer.

Errors
No errors are defined.

Application Usage
Before performing the first refresh of a subwindow, portable applications should call touchwin() or touchline() on the parent window.

Each window maintains internal descriptions of the screen image and status. The screen image is shared among all windows in the window hierarchy. Refresh operations rely on information on what has changed within a window, which is private to each window.

Refreshing a window, when updates were made to a different window, may fail to perform needed updates because the windows do not share this information.

A new full-screen window is created by calling:
newwin(0, 0, 0, 0);

See Also
delwin(), is_linetouched(), doupdate(), <curses.h>.

doupdate()

Name
doupdate, refresh, wnoutrefresh, wrefresh - refresh windows and lines

Synopsis
#include <curses.h>
int doupdate(void);
int refresh(void);
int wnoutrefresh(WINDOW *win);
int wrefresh(WINDOW *win);

Description
The refresh() and wrefresh() functions refresh the current or specified window. The functions position the terminal's cursor at the cursor position of the window, except that if the leaveok() mode has been enabled, they may leave the cursor at an arbitrary position.

The wnoutrefresh() function determines which parts of the terminal may need updating. The doupdate() function sends to the terminal the commands to perform any required changes.
Return Value
Upon successful completion, these functions return OK. Otherwise they return ERR.

Errors
No errors are defined.

Application Usage
Refreshing an entire window is typically more efficient than refreshing several subwindows separately. An efficient sequence is to call wnoutrefresh() on each subwindow that has changed, followed by a call to doupdate(), which updates the terminal.

The refresh() or wrefresh() function (or wnoutrefresh() followed by doupdate()) must be called to send output to the terminal, as other Curses functions merely manipulate data structures.

See Also
clearok(), redrawwin(), <curses.h>.

dupwin()

Name
dupwin - duplicate a window

Synopsis
#include <curses.h>

WINDOW *dupwin(WINDOW *win);

Description
The dupwin() function creates a duplicate of the window win.

Return Value
Upon successful completion, dupwin() returns a pointer to the new window. Otherwise, it returns a null pointer.

Errors
No errors are defined.

See Also
derwin(), doupdate(), <curses.h>.

echo()

Name
echo, noecho -- enable/disable terminal echo
Synopsis

```c
#include <curses.h>
int echo(void);
int noecho(void);
```

Description

The `echo()` function enables Echo mode for the current screen. The `noecho()` function disables Echo mode for the current screen. Initially, curses software Echo mode for the current screen is enabled and hardware echo mode of the tty driver is disabled. `echo()` and `noecho()` control software echo only. Hardware echo must remain disabled for the duration of the application, else the behavior is undefined.

Return Value

Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors

No errors are defined.

See Also

`getch()`, `<curses.h>`.

echochar()

Name

echochar, wechochar - echo single-byte character and rendition to a window and refresh

Synopsis

```c
#include <curses.h>
int echochar(const chtype ch);
int wechochar(WINDOW *win, const chtype ch);
```

Description

The `echochar()` function is equivalent to a call to `addch()` followed by a call to `refresh()`.

The `wechochar()` function is equivalent to a call to `waddch()` followed by a call to `wrefresh()`.

Return Value

Upon successful completion, these functions return OK. Otherwise they return ERR.

Errors

No errors are defined.
Application Usage
These functions are only guaranteed to operate reliably on character sets in which each character fits into a single byte, whose attributes can be expressed using only constants with the A_ prefix.

See Also
addch(), douupdate(), echo_wchar(), <curses.h>.

echo_wchar()

Name
echo_wchar, wecho_wchar - write a complex character and immediately refresh the window

Synopsis
#include <curses.h>

int echo_wchar(const cchar_t *wch);
int wecho_wchar(WINDOW *win, const cchar_t *wch);

Description
The echo_wchar() function is equivalent to calling add_wch() and then calling refresh().

The wecho_wchar() function is equivalent to calling wadd_wch() and then calling wrefresh().

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

See Also
addch(), add_wch(), douupdate(), <curses.h>.

endwin()

Name
endwin - suspend Curses session

Synopsis
#include <curses.h>

int endwin(void);
**Description**

The endwin() function restores the terminal after Curses activity by at least restoring the saved shell terminal mode, flushing any output to the terminal and moving the cursor to the first column of the last line of the screen. Refreshing a window resumes program mode. The application must call endwin() for each terminal being used before exiting. If newterm() is called more than once for the same terminal, the first screen created must be the last one for which endwin() is called.

**Return Value**

Upon successful completion, endwin() returns OK. Otherwise, it returns ERR.

**Errors**

No errors are defined.

**Application Usage**

The endwin() function does not free storage associated with a screen, so delscreen() should be called after endwin() if a particular screen is no longer needed.

To leave Curses mode temporarily, portable applications should call endwin(). Subsequently, to return to Curses mode, they should call doupdate(), refresh() or wrefresh().

**See Also**

delscreen(), doupdate(), initscr(), isendwin(), <curses.h>.

---

**erase()**

**Name**

erase, werase - clear a window

**Synopsis**

```c
#include <curses.h>

int erase(void);

int werase(WINDOW *win);
```

**Description**

Refer to clear().

---

**erasechar()**

**Name**

erasechar, erasewchar, killchar, killwchar - terminal environment query functions

**Synopsis**

```c
#include <curses.h>

char erasechar(void);
```
int erasewchar(wchar_t *ch);
char killchar(void);
int killwchar(wchar_t *ch);

Description
The erasechar() function returns the current erase character. The erasewchar() function stores the current erase character in the object pointed to by ch. If no erase character has been defined, the function will fail and the object pointed to by ch will not be changed.

The killchar() function returns the current line kill character. The killwchar() function stores the current line kill character in the object pointed to by ch. If no line kill character has been defined, the function will fail and the object pointed to by ch will not be changed.

Return Value
The erasechar() function returns the erase character and killchar() returns the line kill character. The return value is unspecified when these characters are multi-byte characters.

Upon successful completion, erasewchar() and killwchar() return OK. Otherwise, they return ERR.

Errors
No errors are defined.

Application Usage
The erasechar() and killchar() functions are only guaranteed to operate reliably on character sets in which each character fits into a single byte, whose attributes can be expressed using only constants with the A_ prefix. Moreover, they do not reliably indicate cases in which when the erase or line kill character, respectively, has not been defined. The erasewchar() and killwchar() functions overcome these limitations.

See Also
clearok(), delscreen(), tcgetattr(), <curses.h>.

filter()

Name
filter - disable use of certain terminal capabilities

Synopsis
#include <curses.h>
void filter(void);
Description
The filter() function changes the algorithm for initializing terminal capabilities that assume that the terminal has more than one line. A subsequent call to initscr() or newterm() performs the following additional actions:

- Disable use of clear, cud, cud1, cup, cuu1 and vpa
- Set the value of the home string to the value of the cr string
- Set lines equal to 1.

Any call to filter() must precede the call to initscr() or newterm().

Return Value
The filter() function does not return a value.

Errors
No errors are defined.

See Also
initscr(), <curses.h>.

---

flash()

Name
flash - flash the screen

Synopsis
#include <curses.h>

int flash(void);

Description
The flash() function alerts the user. It flashes the screen, or if that is not possible, it sounds the audible alarm on the terminal. If neither signal is possible, nothing happens.

Return Value
The flash() function always returns OK.

Errors
No errors are defined.

Application Usage
Nearly all terminals have an audible alarm, but only some can flash the screen.

See Also
beep(), <curses.h>
flushinp()

Name
flushinp - discard input

Synopsis
#include <curses.h>

int flushinp(void);

Description
The flushinp() function discards (flushes) any characters in the input buffer associated with the current screen.

Return Value
The flushinp() function always returns OK.

Errors
No errors are defined.

See Also
<curses.h>.

getbegyx()

Name
gotbegyx, getmaxyx, getparyx, getyx - get cursor and window coordinates

Synopsis
#include <curses.h>

void getbegyx(WINDOW *win, int y, int x);
void getmaxyx(WINDOW *win, int y, int x);
void getparyx(WINDOW *win, int y, int x);
void getyx(WINDOW *win, int y, int x);

Description
The getyx() macro stores the cursor position of the specified window in y and x.

The getparyx() macro, if the specified window is a subwindow, stores in y and x the coordinates of the window’s origin relative to its parent window. Otherwise, -1 is stored in y and x.

The getbegyx() macro stores the absolute screen coordinates of the specified window’s origin in y and x.

The getmaxyx() macro stores the number of rows of the specified window in y and stores the window’s number of columns in x.
Return Value
No return values are defined.

Errors
No errors are defined.

Application Usage
These interfaces are macros and ‘&’ cannot be used before the y and x arguments. Traditional implementations have often defined the following macros:

void getbegx(WINDOW *win, int x);
void getbegy(WINDOW *win, int y);
void getmaxx(WINDOW *win, int x);
void getmaxy(WINDOW *win, int y);
void getparx(WINDOW *win, int x);
void getpary(WINDOW *win, int y);

Although getbegyx(), getmaxyx() and getparyx() provide the required functionality, this does not preclude applications from defining these macros for their own use. For example, to implement void getbegx(WINDOW *win, int x); the macro would be

#define getbegx(_win,_x); /*
  { /*
   _y; /*
   getbegyx(_win,_y,_x); /*
  }

See Also
<curses.h>

getbkgd()

Name
getbkgd - get background character and rendition using a single-byte character

Synopsis
#include <curses.h>

cttype getbkgd(WINDOW *win);

Description
Refer to bkgd().

getbkgrnd()

Name
getbkgrnd - get background character and rendition

Synopsis
#include <curses.h>

int getbkgrnd(cchar_t *ch);
**getcchar()**

**Name**
getcchar - get a wide character string and rendition from a cchar_t

**Synopsis**
```
#include <curses.h>

int getcchar(const cchar_t *wcval, wchar_t *wch, attr_t *attrs,
             short *color_pair, void *opts);
```

**Description**
When `wch` is not a null pointer, the `getcchar()` function extracts information from a cchar_t defined by `wcval`, stores the character attributes in the object pointed to by `attrs`, stores the color pair in the object pointed to by `color_pair`, and stores the wide character string referenced by `wcval` into the array pointed to by `wch`.

When `wch` is a null pointer, `getcchar()` obtains the number of wide characters in the object pointed to by `wcval` and does not change the objects pointed to by `attrs` or `color_pair`.

The `opts` argument is reserved for definition in a future edition of this document. Currently, the application must provide a null pointer as `opts`.

**Return Value**
When `wch` is a null pointer, `getcchar()` returns the number of wide characters referenced by `wcval`, including the null terminator.

When `wch` is not a null pointer, `getcchar()` returns OK upon successful completion, and ERR otherwise.

**Errors**
No errors are defined.

**Application Usage**
The `wcval` argument may be a value generated by a call to `setcchar()` or by a function that has a cchar_t output argument. If `wcval` is constructed by any other means, the effect is unspecified.

**See Also**
`attroff()`, `can_change_color()`, `setcchar()`, `<curses.h>`.

**getch()**

**Name**
getch, wgetch, mvgetch, mvwgetch - get a single-byte character from the terminal
Synopsis

#include <curses.h>

int getch(void);
int mvgetch(int y, int x);
int mvwgetch(WINDOW *win, int y, int x);
int wgetch(WINDOW *win);

Description

These functions read a single-byte character from the terminal associated with the current or specified window. The results are unspecified if the input is not a single-byte character. If keypad() is enabled, these functions respond to the pressing of a function key by returning the corresponding KEY_ value defined in <curses.h>.

If echoing is enabled, then the character is echoed as though it were provided as an input argument to addch(), except for the following characters:

- `<backspace>`: The input is interpreted and then the character at the resulting cursor position is deleted as though delch() were called, except that if the cursor was originally in the first column of the line, then the user is alerted as though beep() were called.
- `<left-arrow>`: The user is alerted as though beep() were called. Information concerning the function keys is not returned to the caller.
- `Function keys`: The user is alerted as though beep() were called. Information concerning the function keys is not returned to the caller.

If the current or specified window is not a pad, and it has been moved or modified since the last refresh operation, then it will be refreshed before another character is read.

Return Value

Upon successful completion, `getch()`, `mvgetch()`, `mvwgetch()` and `wgetch()` return the single-byte character, KEY_ value, or ERR. When in the nodelay mode and no data is available, ERR is returned.

Errors

No errors are defined.

Application Usage

Applications should not define the escape key by itself as a single-character function.

When using these functions, nocbreak mode (nocbreak()) and echo mode (echo()) should not be used at the same time. Depending on the state of the terminal when each character is typed, the program may produce undesirable results.

See Also

cbreak(), doupdate(), insch(), <curses.h>.
getmaxyx()

Name
getmaxyx - get size of a window

Synopsis
#include <curses.h>

void getmaxyx(WINDOW *win, int y, int x);

Description
Refer to getbegyx().

getnstr()

Name
getnstr, getstr, mvgetnstr, mvgetstr, mvwgetnstr, mvwgetstr, wgetstr, wgetnstr - get a multi-byte character string from the terminal

Synopsis
#include <curses.h>

int getnstr(char *str, int n);
int getstr(char *str);
int mvgetnstr(int y, int x, char *str, int n);
int mvgetstr(int y, int x, char *str);
int mvwgetnstr(WINDOW *win, int y, int x, char *str, int n);
int mvwgetstr(WINDOW *win, int y, int x, char *str);
int wgetnstr(WINDOW *win, char *str, int n);
int wgetstr(WINDOW *win, char *str);

Description
The effect of getstr() is as though a series of calls to getch() were made, until a newline or carriage return is received. The resulting value is placed in the area pointed to by str. The string is then terminated with a null byte. The getnstr(), mvgetnstr(), mvwgetnstr() and wgetnstr() functions read at most n bytes, thus preventing a possible overflow of the input buffer. The user’s erase and kill characters are interpreted, as well as any special keys (such as function keys, home key, clear key, and so on).

The mvgetstr() function is identical to getstr() except that it is as though it is a call to move() and then a series of calls to getch(). The mvwgetstr() function is identical to getstr() except it is as though a call to wmove() is made and then a series of calls to wgetch(). The mvwgetnstr() function is identical to getnstr() except that it is as though it is a call to move() and then a series of calls to getch(). The mvwgetnstr() function is identical to getstr() except it is as though a call to wmove() is made and then a series of calls to wgetch().
The getnstr(), wgetnstr(), mvgetnstr() and mvwgetnstr() functions will only return the entire multi-byte sequence associated with a character. If the array is large enough to contain at least one character, the functions fill the array with complete characters. If the array is not large enough to contain any complete characters, the function fails.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

Application Usage
Reading a line that overflows the array pointed to by str with getstr(), mvgetstr(), mvwgetstr() or wgetstr() causes undefined results. The use of getnstr(), mvgetnstr(), mvwgetnstr() or wgetnstr(), respectively, is recommended.

See Also
beep(), getch(), <curses.h>.

gtn_wstr()  

Name
gtn_wstr, get_wstr, mvgetn_wstr, mvget_wstr, mvwgetn_wstr, mvwget_wstr, wgetn_wstr, wget_wstr - get an array of wide characters and function key codes from a terminal

Synopsis
#include <curses.h>

int getn_wstr(wint_t *wstr, int n);
int get_wstr(wint_t *wstr);
int mvgetn_wstr(int y, int x, wint_t *wstr, int n);
int mvget_wstr(int y, int x, wint_t *wstr);
int mvwgetn_wstr(WINDOW *win, int y, int x, wint_t *wstr, int n);
int mvwget_wstr(WINDOW *win, int y, int x, wint_t *wstr);
int wgetn_wstr(WINDOW *win, wint_t *wstr, int n);
int wget_wstr(WINDOW *win, wint_t *wstr);

Description
The effect of get_wstr() is as though a series of calls to get_wch() were made, until a newline character, end-of-line character, or end-of-file character is processed. An end-of-file character is represented by WEOF, as defined in <wchar.h>. A newline or end-of-line is represented as its wchar_t value. In all instances, the end of the string is terminated by a null wchar_t. The resulting values are placed in the area pointed to by wstr.
The user's erase and kill characters are interpreted and affect the sequence of characters returned.

The effect of wget_wstr() is as though a series of calls to wget_wch() were made.

The effect of mvget_wstr() is as though a call to move() and then a series of calls to get_wch() were made. The effect of mvwget_wstr() is as though a call to wmove() and then a series of calls to wget_wch() were made. The effect of mvwget_nwstr() is as though a call to wmove() and then a series of calls to wget_wch() were made.

The getn_wstr(), mvgetn_wstr(), mvwgetn_wstr() and wgetn_wstr() functions read at most \( n \) characters, letting the application prevent overflow of the input buffer.

**Return Value**

Upon successful completion, these functions return OK. Otherwise, they return ERR.

**Errors**

No errors are defined.

**Application Usage**

Reading a line that overflows the array pointed to by wstr with wget_wstr(), mvwget_wstr(), mvwget_nwstr() or wget_nwstr() causes undefined results. The use of getn_wstr(), mvgetn_wstr(), mvwgetn_wstr() or wgetn_wstr(), respectively, is recommended.

These functions cannot return KEY_ values as there is no way to distinguish a KEY_ value from a valid wchar_t value.

**See Also**

get_wch(), getstr(), <curses.h>, <wchar.h>.

---

**getparyx()**

**Name**

getparyx - get subwindow origin coordinates

**Synopsis**

```c
#include <curses.h>

void getparyx(WINDOW *win, int y, int x);
```

**Description**

Refer to getbegyx().

**getstr()**

**Name**

getstr - get a multi-byte character string from the terminal
Synopsis

#include <curses.h>

int getstr(char *str);

Description

Refer to getnstr().

get_wch()

Name
get_wch, mvget_wch, mvwget_wch, wget_wch - get a wide character from a terminal

Synopsis

#include <curses.h>

int get_wch(wint_t *ch);

int mvget_wch(int y, int x, wint_t *ch);

int mvwget_wch(WINDOW *win, int y, int x, wint_t *ch);

int wget_wch(WINDOW *win, wint_t *ch);

Description

These functions read a character from the terminal associated with the current or specified window. If keypad() is enabled, these functions respond to the pressing of a function key by setting the object pointed to by ch to the corresponding KEY_value defined in <curses.h> and returning KEY_CODE_YES.

Processing of terminal input is subject to the general rules.

If echoing is enabled, then the character is echoed as though it were provided as an input argument to add_wch(), except for the following characters:

- <backspace>, <left-arrow> and the current erase character:
  The input is interpreted and then the character at the resulting cursor position is deleted as though delch() were called, except that if the cursor was originally in the first column of the line, then the user is alerted as though beep() were called.

- Function keys
  The user is alerted as though beep() were called. Information concerning the function keys is not returned to the caller.

If the current or specified window is not a pad, and it has been moved or modified since the last refresh operation, then it will be refreshed before another character is read.

Return Value

When these functions successfully report the pressing of a function key, they return KEY_CODE_YES. When they successfully report a wide character, they return OK. Otherwise, they return ERR.
Errors
No errors are defined.

Application Usage
Applications should not define the escape key by itself as a single-character function.

When using these functions, nocbreak mode and echo mode should not be used at the same time. Depending on the state of the terminal when each character is typed, the application may produce undesirable results.

See Also
`beep()`, `cbreak()`, `ins_wch()`, `keypad()`, `move()`, `<curses.h>`, `<wchar.h>`.

getwin()

Name
getwin, putwin - dump window to, and reload window from, a file

Synopsis
```c
#include <curses.h>
WINDOW *getwin(FILE *filep);
int putwin(WINDOW *win, FILE *filep);
```

Description
The `getwin()` function reads window-related data stored in the file by `putwin()`. The function then creates and initializes a new window using that data.

The `putwin()` function writes all data associated with `win` into the stdio stream to which `filep` points, using an unspecified format. This information can be retrieved later using `getwin()`.

Return Value
Upon successful completion, `getwin()` returns a pointer to the window it created. Otherwise, it returns a null pointer.

Upon successful completion, `putwin()` returns OK. Otherwise, it returns ERR.

Errors
No errors are defined.

See Also
`scr_dump()`, `<curses.h>`.

get_wstr()

Name
get_wstr - get an array of wide characters and function key codes from a terminal
Synopsis
#include <curses.h>
int get_wstr(wint_t *wstr);

Description
Refer to getn_wstr().

goingx()

Name
getyx - get cursor coordinates

Synopsis
#include <curses.h>
void getyx(WINDOW *win, int y, int x);

Description
Refer to getbegyx().

halfdelay()

Name
halfdelay - control input character delay mode

Synopsis
#include <curses.h>
int halfdelay(int tenths);

Description
The halfdelay() function sets the input mode for the current window to Half-Delay Mode and specifies tenths of seconds as the half-delay interval. The tenths argument must be in a range from 1 up to and including 255.

Return Value
Upon successful completion, halfdelay() returns OK. Otherwise, it returns ERR.

Errors
No errors are defined.

Application Usage
The application can call nocbreak() to leave Half-Delay mode.

See Also
cbreak(), <curses.h>.
has_colors()

Name
has_colors - indicate whether terminal supports colors

Synopsis
#include <curses.h>

bool has_colors(void);

Description
Refer to can_change_color().

has_ic()

Name
has_ic, has_il - query functions for terminal insert and delete capability

Synopsis
#include <curses.h>

bool has_ic(void);
bool has_il(void);

Description
The has_ic() function indicates whether the terminal has insert- and delete-character capabilities.

The has_il() function indicates whether the terminal has insert- and delete-line capabilities, or can simulate them using scrolling regions.

Return Value
The has_ic() function returns TRUE if the terminal has insert- and delete-character capabilities. Otherwise, it returns FALSE.

The has_il() function returns TRUE if the terminal has insert- and delete-line capabilities. Otherwise, it returns FALSE.

Errors
No errors are defined.

Application Usage
The has_il() function may be used to determine if it would be appropriate to turn on physical scrolling using scrollok().

See Also
<curses.h>.
hline()

Name
hline, mvhline, mvvline, mvwhline, mvwvline, vline, whline, wvline - draw lines from single-byte characters and renditions

Synopsis
```
#include <curses.h>

int hline(chtype ch, int n);
int mvhline(int y, int x, chtype ch, int n);
int mvvline(int y, int x, chtype ch, int n);
int mvwhline(WINDOW *win, int y, int x, chtype ch, int n);
int mvwvline(WINDOW *win, int y, int x, chtype ch, int n);
int vline(chtype ch, int n);
int whline(WINDOW *win, chtype ch, int n);
int wvline(WINDOW *win, chtype ch, int n);
```

Description
These functions draw a line in the current or specified window starting at the current or specified position, using \textit{ch}. The line is at most \textit{n} positions long, or as many as fit into the window.

These functions do not advance the cursor position. These functions do not perform special character processing. These functions do not perform wrapping.

The hline(), mvhline(), mvwhline() and whline() functions draw a line proceeding toward the last column of the same line.

The vline(), mvvline(), mvwvline() and wvline() functions draw a line proceeding toward the last line of the window.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

hline()

Application Usage
These functions are only guaranteed to operate reliably on character sets in which each character fits into a single byte, whose attributes can be expressed using only constants with the A\_ prefix.
**hline_set()**

**Name**

hline_set, mvhline_set, mvvline_set, mvwhline_set, mvwvline_set, vline_set, whline_set, wvline_set - draw lines from complex characters and renditions

**Synopsis**

```c
#include <curses.h>

int hline_set(const cchar_t *wch, int n);
int mvhline_set(int y, int x, const cchar_t *wch, int n);
int mvvline_set(int y, int x, const cchar_t *wch, int n);
int mvwhline_set(WINDOW *win, int y, int x, const cchar_t *wch, int n);
int mvwvline_set(WINDOW *win, int y, int x, const cchar_t *wch, int n);
int vline_set(const cchar_t *wch, int n);
int whline_set(WINDOW *win, const cchar_t *wch, int n);
int wvline_set(WINDOW *win, cchar_t *const wch, int n);
```

**Description**

These functions draw a line in the current or specified window starting at the current or specified position, using ch. The line is at most n positions long, or as many as fit into the window.

These functions do not advance the cursor position. These functions do not perform special character processing. These functions do not perform wrapping.

The hline_set(), mvhline_set(), mvvline_set() and whline_set() functions draw a line proceeding toward the last column of the same line.

The vline_set(), mvvline_set(), mvvvline_set() and wvline_set() functions draw a line proceeding toward the last line of the window.

**Return Value**

Upon successful completion, these functions return OK. Otherwise, they return ERR.

**Errors**

No errors are defined.

**See Also**

border_set(), box(), hline_set(), <curses.h>.
idcok()

Name
idcok - enable or disable use of hardware insert- and delete-character features

Synopsis
#include <curses.h>
void idcok(WINDOW *win, bool bf);

Description
The idcok() function specifies whether the implementation may use hardware
insert- and delete-character features in win if the terminal is so equipped. If bf is
TRUE, use of these features in win is enabled. If bf is FALSE, use of these features
in win is disabled. The initial state is TRUE.

Return Value
The idcok() function does not return a value.

Errors
No errors are defined.

See Also
clearok(), doupdate(), <curses.h>.

idlok()

Name
idlok - enable or disable use of terminal insert- and delete-line features

Synopsis
#include <curses.h>
int idlok(WINDOW *win, bool bf);

Description
Refer to clearok().

immedok()

Name
immedok - enable or disable immediate terminal refresh

Synopsis
#include <curses.h>
void immedok(WINDOW *win, bool bf);
Enhanced Curses

**Description**
The immedok() function specifies whether the screen is refreshed whenever the window pointed to by win is changed. If bf is TRUE, the window is implicitly refreshed on each such change. If bf is FALSE, the window is not implicitly refreshed. The initial state is FALSE.

**Return Value**
The immedok() function does not return a value.

**Errors**
No errors are defined.

**Application Usage**
The immedok() function is useful for windows that are used as terminal emulators.

**See Also**
clearok(), doupdate(), <curses.h>.

---

**inch()**

**Name**
inch, mvinch, mvwinch, winch - input a single-byte character and rendition from a window

**Synopsis**
```
#include <curses.h>

chtype inch(void);
chtype mvinch(int y, int x);
chtype mvwinch(WINDOW *win, int y, int x);
chtype winch(WINDOW *win);
```

**Description**
These functions return the character and rendition, of type chtype, at the current or specified position in the current or specified window.

**Return Value**
Upon successful completion, the functions return the specified character and rendition. Otherwise, they return (chtype)ERR.

**Errors**
No errors are defined.

**Application Usage**
These functions are only guaranteed to operate reliably on character sets in which each character fits into a single byte, whose attributes can be expressed using only constants with the A_ prefix.
See Also

<curses.h>.

inchnstr()

Name

inchnstr, inchstr, mvchinexpr, mvchestr, mvwinchinstr, mvwinchestr, winchinstr,
winchestr - input an array of single-byte characters and renditions from a window

Synopsis

#include <curses.h>

int inchnstr(chr *chstr, int n);
int inchstr(chr *chstr);
int mvchinexpr(int y, int x, chr *chstr, int n);
int mvchestr(int y, int x, chr *chstr);
int mvwinchinexpr(WINDOW *win, int y, int x, chr *chstr, int n);
int mvchestr(WINDOW *win, int y, int x, chr *chstr);
int winchinstr(WINDOW *win, chr *chstr, int n);
int winchestr(WINDOW *win, chr *chstr);

Description

These functions place characters and renditions from the current or specified
window into the array pointed to by chstr, starting at the current or specified
position and ending at the end of the line.

The inchnstr(), mvchinexpr(), mvchestr() and winchinstr() functions store at
most n elements from the current or specified window into the array pointed to by
chstr.

Return Value

Upon successful completion, these functions return OK. Otherwise, they return
ERR.

Errors

No errors are defined.

Application Usage

Reading a line that overflows the array pointed to by chstr with inchstr(),
mvchinexpr(), mvchestr() or winchinstr() causes undefined results. The use of
inchnstr(), mvchinexpr(), mvchestr() or winchinstr(), respectively, is
recommended.

See Also

inch(), <curses.h>.
init_color()

Name
init_color, init_pair - redefine specified color or color pair

Synopsis
#include <curses.h>

int init_color(short color, short red, short green, short blue);

int init_pair(short pair, short f, short b);

Description
Refer to can_change_color().

initscr()

Name
initscr, newterm - screen initialization functions

Synopsis
#include <curses.h>

WINDOW *initscr(void);

SCREEN *newterm(char *type, FILE *outfile, FILE *infile);

Description
The initscr() function determines the terminal type and initializes all implementation data structures. The TERM environment variable specifies the terminal type. The initscr() function also causes the first refresh operation to clear the screen. If errors occur, initscr() writes an appropriate error message to standard error and exits. The only functions that can be called before initscr() or newterm() are filter(), ripoffline(), slk_init(), use_env() and the functions whose prototypes are defined in <term.h>. Portable applications must not call initscr() twice.

The newterm() function can be called as many times as desired to attach a terminal device. The type argument points to a string specifying the terminal type, except that if type is a null pointer, the TERM environment variable is used. The outfile and infile arguments are file pointers for output to the terminal and input from the terminal, respectively. It is unspecified whether Curses modifies the buffering mode of these file pointers. The newterm() function should be called once for each terminal.

The initscr() function is equivalent to:
newterm(getenv("TERM"), stdout, stdin);
return stdscr;

If the current disposition for the signals SIGINT, SIGQUIT or SIGTSTP is SIGDFL, then initscr() may also install a handler for the signal, which may remain in effect for the life of the process or until the process changes the disposition of the signal.

The initscr() and newterm() functions initialize the cur_term external variable.
**innstr()**

**Name**

innstr, instr, mvinnstr, mvinstr, mvwinnstr, mvwinstr, winnstr, winstr - input a multi-byte character string from a window

**Synopsis**

```
#include <curses.h>

int innstr(char *str, int n);
int instr(char *str);
int mvinnstr(int y, int x, char *str, int n);
int mvinstr(int y, int x, char *str);
int mvwinnstr(WINDOW *win, int y, int x, char *str, int n);
int mvwinstr(WINDOW *win, int y, int x, char *str);
int winnstr(WINDOW *win, char *str, int n);
int winstr(WINDOW *win, char *str);
```

**Description**

These functions place a string of characters from the current or specified window into the array pointed to by str, starting at the current or specified position and ending at the end of the line.
The `innstr()`, `mvinnstr()`, `mvwinnstr()` and `winnstr()` functions store at most \( n \) bytes in the string pointed to by `str`.

The `innstr()`, `mvinnstr()`, `mvwinnstr()` and `winnstr()` functions will only store the entire multi-byte sequence associated with a character. If the array is large enough to contain at least one character the array is filled with complete characters. If the array is not large enough to contain any complete characters, the function fails.

**Return Value**

Upon successful completion, `instr()`, `mvinstr()`, `mvwinnstr()` and `winnstr()` return OK.

Upon successful completion, `innstr()`, `mvinnstr()`, `mvwinnstr()` and `winnstr()` return the number of characters actually read into the string. Otherwise, all these functions return ERR.

**Errors**

No errors are defined.

**Application Usage**

Since multi-byte characters may be processed, there might not be a one-to-one correspondence between the number of column positions on the screen and the number of bytes returned.

These functions do not return rendition information.

Reading a line that overflows the array pointed to by `str` with `instr()`, `mvinstr()`, `mvwinnstr()` or `winnstr()` causes undefined results. The use of `innstr()`, `mvinnstr()`, `mvwinnstr()` or `winnstr()`, respectively, is recommended.

**See Also**

`<curses.h>`.

innwstr()
int innwstr(WINDOW *win, wchar_t *wstr, int n);
int winnwstr(WINDOW *win, wchar_t *wstr);

Description
These functions place a string of wchar_t characters from the current or specified
window into the array pointed to by wstr starting at the current or specified cursor
position and ending at the end of the line.

These functions will only store the entire wide character sequence associated with
a spacing complex character. If the array is large enough to contain at least one
complete spacing complex character, the array is filled with complete characters. If
the array is not large enough to contain any complete characters this is an error.

The innwstr(), mvinnwstr(), mvwinnwstr() and winnwstr() functions store at most
n characters in the array pointed to by wstr.

Return Value
Upon successful completion, innwstr(), mvinwstr(), mvwinwstr() and winwstr()
return OK.

Upon successful completion, innwstr(), mvinnwstr(), mvwinnwstr() and winnwstr()
return the number of characters actually read into the string. Otherwise, all these
functions return ERR.

Errors
No errors are defined.

Application Usage
Reading a line that overflows the array pointed to by wstr with innwstr(),
mvinnwstr(), mvwinwstr() or winwstr() causes undefined results. The use of
innwstr(), mvinnwstr(), mvwinnwstr() or winnwstr(), respectively, is recommended.

These functions do not return rendition information.

See Also
<curses.h>.

insch()
Description
These functions insert the character and rendition from \textit{ch} into the current or specified window at the current or specified position.

These functions do not perform wrapping. These functions do not advance the cursor position. These functions perform special-character processing, with the exception that if a newline is inserted into the last line of a window and scrolling is not enabled, the behavior is unspecified.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

Application Usage
These functions are only guaranteed to operate reliably on character sets in which each character fits into a single byte, whose attributes can be expressed using only constants with the \texttt{A_} prefix.

See Also
\textit{ins\_wch() <curses.h>}

\textbf{insdelln()}

Name
insdelln, winsdelln - delete or insert lines into a window

Synopsis
\begin{verbatim}
#include <curses.h>

int insdelln(int n);

int winsdelln(WINDOW *win, int n);
\end{verbatim}

Description
The \texttt{insdelln()} and \texttt{windsdelln()} functions perform the following actions:
\begin{itemize}
  \item If \texttt{n} is positive, these functions insert \texttt{n} lines into the current or specified window before the current line. The \texttt{n} last lines are no longer displayed.
  \item If \texttt{n} is negative, these functions delete \texttt{n} lines from the current or specified window starting with the current line, and move the remaining lines toward the cursor. The last \texttt{n} lines are cleared.
\end{itemize}

The current cursor position remains the same.
Return Value

Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors

No errors are defined.

See Also

deleteln(), insertln(), <curses.h>.

insertln()  

Name

insertln, wininsertln - insert lines into a window

Synopsis

#include <curses.h>

int insertln(void);
int wininsertln(WINDOW *win);

Description

The insertln() and wininsertln() functions insert a blank line before the current line in the current or specified window. The bottom line is no longer displayed. The cursor position does not change.

Return Value

Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors

No errors are defined.

See Also

insdelln(), <curses.h>.

insnstr()  

Name

insnstr, insstr, mvinsnstr, mvinsstr, mvwinsnstr, mvwinsstr, winsnstr, winsstr - insert a multi-byte character string into a window

Synopsis

#include <curses.h>

int insnstr(const char *str, int n);
int insstr(const char *str);
int mvinsnstr(int y, int x, const char *str, int n);
int mvinsstr(int y, int x, const char *str);
int mvwinsnstr(WINDOW *win, int y, int x, const char *str, int n);
int mvwinsstr(WINDOW *win, int y, int x, const char *str);
int winsnstr(WINDOW *win, const char *str, int n);
int winsstr(WINDOW *win, const char *str);

Description
These functions insert a character string (as many characters as will fit on the line) before the current or specified position in the current or specified window.

These functions do not advance the cursor position. These functions perform special-character processing. The innstr() and innwstr() functions perform wrapping. The instr() and () inswstr functions do not perform wrapping.

The insnstr(), mvinsnstr(), mvwinsnstr() and winsnstr() functions insert at most \( n \) bytes. If \( n \) is less than 1, the entire string is inserted.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

Application Usage
Since the string may contain multi-byte characters, there might not be a one-to-one correspondence between the number of column positions occupied by the characters and the number of bytes in the string.

See Also
<stdio.h>

---

ins_nwstr()  

Name
ins_nwstr, ins_wstr, mvins_nwstr, mvins_wstr, mvwins_nwstr, mvwins_wstr, wins_nwstr, wins_wstr - insert a wide-character string into a window

Synopsis
#include <curses.h>

int ins_nwstr(const wchar_t *wstr, int n);
int ins_wstr(const wchar_t *wstr);
int mvins_nwstr(int y, int x, const wchar_t *wstr, int n);
int mvins_wstr(int y, int x, const wchar_t *wstr);
int mvwins_nwstr(WINDOW *win, int y, int x, const wchar_t *wstr, int n);

---
int mvwins_wstr(WINDOW *win, int y, int x, const wchar_t *wstr);
int wins_nwstr(WINDOW *win, const wchar_t *wstr, int n);
int wins_wstr(WINDOW *win, const wchar_t *wstr);

Description
These functions insert a wchar_t character string (as many wchar_t characters as will fit on the line) in the current or specified window immediately before the current or specified position.

Any non-spacing characters in the string are associated with the first spacing character in the string that precedes the non-spacing characters. If the first character in the string is a non-spacing character, these functions will fail.

These functions do not perform wrapping. These functions do not advance the cursor position. These functions perform special-character processing.

The ins_nwstr(), mvins_nwstr(), mvwins_nwstr() and wins_nwstr() functions insert at most n wchar_t characters. If n is less than 1, then the entire string is inserted.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

See Also
<stdio.h>.

insstr()

Name
insstr - insert a multi-byte character string into the current window

Synopsis
#include <curses.h>
int insstr(const char *str);

Description
Refer to insnstr().

instr()

Name
instr - input a multi-byte character string from the current window
Synopsis
#include <curses.h>
int instr(char *str);

Description
Refer to innstr().

ins_wch()

Name
ins_wch, mvins_wch, mvwins_wch, wins_wch - insert a complex character and
rendition into a window

Synopsis
#include <curses.h>
int ins_wch(const cchar_t *wch);
int wins_wch(WINDOW *win, const cchar_t *wch);
int mvins_wch(int y, int x, const cchar_t *wch);
int mvwins_wch(WINDOW *win, int y, int x, const cchar_t *wch);

Description
These functions insert the complex character wch with its rendition in the current
or specified window at the current or specified cursor position.

These functions do not perform wrapping. These functions do not advance the
cursor position. These functions perform special-character processing.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return
ERR.

Errors
No errors are defined.

Application Usage
For non-spacing characters, add_wch() can be used to add the non-spacing
characters to a spacing complex character already in the window.

See Also
add_wch(), <curses.h>.

ins_wstr()

Name
ins_wstr - insert a wide-character string into the current window
Synopsis

```c
#include <curses.h>

int ins_wstr(const wchar_t *wstr);
```

Description

Refer to ins_nwstr().

intrflush()

Name

intrflush - enable or disable flush on interrupt

Synopsis

```c
#include <curses.h>

int intrflush(WINDOW *win, bool bf);
```

Description

The intrflush() function specifies whether pressing an interrupt key (interrupt, suspend or quit) will flush the input buffer associated with the current screen. If `bf` is a boolean that specifies whether pressing an interrupt key (interrupt, suspend or quit) will flush the output buffer associated with the current screen. The default for the option is inherited from the display driver settings. The `win` argument is ignored.

Return Value

Upon successful completion, intrflush() returns OK. Otherwise, it returns ERR.

Errors

No errors are defined.

Application Usage

The same effect is achieved outside Curses using the NOFLSH local mode flag specified in the XBD specification (General Terminal Interface).

See Also

`<curses.h>`.

in_wch()

Name

in_wch, mvin_wch, mvwin_wch, win_wch - input a complex character and rendition from a window

Synopsis

```c
#include <curses.h>

int in_wch(cchar_t *wcval);
```
Description
These functions extract the complex character and rendition from the current or specified position in the current or specified window into the object pointed to by wcval.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

See Also
<curses.h>.

in_wchnstr()

Name
in_wchnstr, in_wchstr, mvin_wchnstr, mwin_wchnstr, mvwin_wchnstr, mvwint_wchnstr, win_wchnstr
win_wchstr - input an array of complex characters and renditions from a window

Synopsis
#include <curses.h>

int in_wchnstr(cchar_t *wchstr, int n);
int in_wchstr(cchar_t *wchstr);
int mvin_wchnstr(int y, int x, cchar_t *wchstr, int n);
int mvin_wchstr(int y, int x, cchar_t *wchstr);
int mvwin_wchnstr(WINDOW *win, int y, int x, cchar_t *wchstr, int n);
int mvwin_wchstr(WINDOW *win, int y, int x, cchar_t *wchstr);
int win_wchnstr(WINDOW *win, cchar_t *wchstr, int n);
int win_wchstr(WINDOW *win, cchar_t *wchstr);

Description
These functions extract characters from the current or specified window, starting at the current or specified position and ending at the end of the line, and place them in the array pointed to by wchstr.

The in_wchnstr(), mvin_wchnstr(), mvwin_wchnstr() and win_wchnstr() fill the array with at most n cchar_t elements.
Return Value

Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors

No errors are defined.

Application Usage

Reading a line that overflows the array pointed to by wchstr with in_wchstr(), mvin_wchstr(), mvwin_wchstr() or win_wchstr() causes undefined results. The use of in_wchnstr(), mvin_wchnstr(), mvwin_wchnstr() or win_wchnstr(), respectively, is recommended.

See Also

in_wch(), <curses.h>.

inwstr()

Name

inwstr - input a string of wide characters from the current window

Synopsis

#include <curses.h>

int inwstr(wchar_t *wstr);

Description

Refer to innwstr().

isendwin()

Name

isendwin - determine whether a screen has been refreshed

Synopsis

#include <curses.h>

bool isendwin(void);

Description

The isendwin() function indicates whether the screen has been refreshed since the last call to endwin().

Return Value

The isendwin() function returns TRUE if endwin() has been called without any subsequent refresh. Otherwise, it returns FALSE.

Errors

No errors are defined.
See Also

dcendwin(), <curses.h>.

is_linetouched()

Name

is_linetouched, is_wintouched, touchline, touchwin, untouchwin, wtouchnl -
window refresh control functions

Synopsis

#include <curses.h>

bool is_linetouched(WINDOW *win, int line);
bool is_wintouched(WINDOW *win);
int touchline(WINDOW *win, int start, int count);
int touchwin(WINDOW *win);
int untouchwin(WINDOW *win);
int wtouchnl(WINDOW *win, int y, int n, int changed);

Description

The touchwin() function touches the specified window (that is, marks it as having
changed more recently than the last refresh operation). The touchline() function
only touches count lines, beginning with line start.

The untouchwin() function marks all lines in the window as unchanged since the
last refresh operation.

Calling wtouchnl(), if changed is 1, touches n lines in the specified window,
starting at line y. If changed is 0, wtouchnl() marks such lines as unchanged since
the last refresh operation.

The is_wintouched() function determines whether the specified window is touched.
The is_linetouched() function determines whether line line of the specified window
is touched.

Return Value

The is_linetouched() and is_wintouched() functions return TRUE if any of the
specified lines, or the specified window, respectively, has been touched since the
last refresh operation. Otherwise, they return FALSE.

Upon successful completion, the other functions return OK. Otherwise, they return
ERR. Exceptions to this are noted in the preceding function descriptions.

Errors

No errors are defined.
Application Usage

Calling touchwin() or touchline() is sometimes necessary when using overlapping windows, since a change to one window affects the other window, but the records of which lines have been changed in the other window do not reflect the change.

See Also

doupdate(), <curses.h>.

keyname()

Name

keyname, key_name - get name of key

Synopsis

#include <curses.h>

char *keyname(int c);

char *key_name(wchar_t c);

Description

The keyname() and key_name() functions generate a character string whose value describes the key \( c \). The \( c \) argument of keyname() can be an 8-bit character or a key code. The \( c \) argument of key_name() must be a wide character.

The string has a format according to the first applicable row in the following table:

<table>
<thead>
<tr>
<th>Input</th>
<th>Format of Returned String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible character</td>
<td>The same character</td>
</tr>
<tr>
<td>Control character</td>
<td>^X</td>
</tr>
<tr>
<td>Meta-character (keyname() only)</td>
<td>M-X</td>
</tr>
<tr>
<td>Key value defined in &lt;curses.h&gt; (keyname() only)</td>
<td>KEY_name</td>
</tr>
<tr>
<td>None of the above</td>
<td>UNKNOWN KEY</td>
</tr>
</tbody>
</table>

The meta-character notation shown above is used only if meta-characters are enabled.

Return Value

Upon successful completion, keyname() returns a pointer to a string as described above. Otherwise, it returns a null pointer.

Errors

No errors are defined.

Application Usage

The return value of keyname() and key_name() may point to a static area which is overwritten by a subsequent call to either of these functions.
Applications normally process meta-characters without storing them into a window. If an application stores meta-characters in a window and tries to retrieve them as wide characters, keyname() cannot detect meta-characters, since wide characters do not support meta-characters.

See Also
meta(), <curses.h>.

kepad()

Name
kepad - enable/disable abbreviation of function keys

Synopsis
#include <curses.h>

int keypad(WINDOW *win, bool bf);

Description
The keypad() function controls keypad translation. If bf is TRUE, keypad translation is turned on. If bf is FALSE, keypad translation is turned off. The initial state is FALSE.

This function affects the behavior of any function that provides keyboard input.

If the terminal in use requires a command to enable it to transmit distinctive codes when a function key is pressed, then after keypad translation is first enabled, the implementation transmits this command to the terminal before an affected input function tries to read any characters from that terminal.

Return Value
Upon successful completion, keypad() returns OK. Otherwise, it returns ERR.

Errors
No errors are defined.

See Also
<curses.h>.

killchar()

Name
killchar, killwchar - terminal environment query functions

Synopsis
#include <curses.h>

char killchar(void);

int killwchar(wchar_t *ch);
Description
Refer to erasechar().

leaveok()

Name
leaveok - control cursor position resulting from refresh operations

Synopsis
#include <curses.h>

int leaveok(WINDOW *win, bool bf);

Description
Refer to clearok().

LINES

Name
LINES - number of lines on terminal screen

Synopsis
#include <curses.h>

extern int LINES;

Description
The external variable LINES indicates the number of lines on the terminal screen.

See Also
initscr(), <curses.h>.

longname()

Name
longname - get verbose description of current terminal

Synopsis
#include <curses.h>

char *longname(void);

Description
The longname() function generates a verbose description of the current terminal. The maximum length of a verbose description is 128 bytes. It is defined only after the call to initscr() or newterm().
Return Value
Upon successful completion, longname() returns a pointer to the description specified above. Otherwise, it returns a null pointer on error.

Errors
No errors are defined.

Application Usage
The return value of longname() may point to a static area which is overwritten by a subsequent call to newterm().

See Also
initscr(), <curses.h>.

meta()

Name
meta - enable/disable meta-keys

Synopsis
#include <curses.h>
int meta(WINDOW *win, bool bf);

Description
Initially, whether the terminal returns 7 or 8 significant bits on input depends on the control mode of the display driver (see the XBD specification, General Terminal Interface). To force 8 bits to be returned, invoke meta(win, TRUE). To force 7 bits to be returned, invoke meta(win, FALSE). The win argument is always ignored. If the terminfo capabilities smm (meta_on) and rmm (meta_off) are defined for the terminal, smm is sent to the terminal when meta(win, TRUE) is called and rmm is sent when meta(win, FALSE) is called.

Return Value
Upon successful completion, meta() returns OK. Otherwise, it returns ERR.

Errors
No errors are defined.

Application Usage
The same effect is achieved outside Curses using the CS7 or CS8 control mode flag specified in the XBD specification (General Terminal Interface).

The meta() function was designed for use with terminals with 7-bit character sets and a “meta” key that could be used to set the eighth bit.

See Also
getch(), <curses.h>.
move()

Name
move, wmove - window cursor location functions

Synopsis
#include <curses.h>

int move(int y, int x);
int wmove(WINDOW *win, int y, int x);

Description
The move() and wmove() functions move the cursor associated with the current or
specified window to \((y, x)\) relative to the window's origin. This function does not
move the terminal’s cursor until the next refresh operation.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return
ERR.

Errors
No errors are defined.

See Also
doupdate(), <curses.h>.

mv

Name
mv - pointer page for functions with mv prefix

Description
Most cases in which a Curses function has the mv prefix \(^1\) indicate that the
function takes \(y\) and \(x\) arguments and moves the cursor to that address as though
move() were first called. (The corresponding functions without the mv prefix
operate at the cursor position.)

The mv prefix is combined with a w prefix to produce Curses functions beginning
with mvw.

The mv and mvw functions are discussed together with the corresponding
functions that do not have these prefixes. They are found on the following entries:

<table>
<thead>
<tr>
<th>Function</th>
<th>Refer to</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvaddch()</td>
<td>mwaddch()</td>
</tr>
<tr>
<td>mvwaddch()</td>
<td>addch()</td>
</tr>
</tbody>
</table>

---

\(^1\) The mvcur(), mvderwin() and mvwin() functions are exceptions to this rule, in that mv is not a prefix with the usual meaning
and there are no corresponding functions without the mv prefix. These functions have entries under their own names.
In the mvprintw() and mvscanw() functions, mv is a prefix with the usual meaning, but the functions have entries under their
own names because the mv function is the first function in the family of functions in alphabetical order.
## Curses

<table>
<thead>
<tr>
<th>Function</th>
<th>Refer to</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvaddchnstr()</td>
<td>addchstr()</td>
</tr>
<tr>
<td>mvaddchstr()</td>
<td>addchstr()</td>
</tr>
<tr>
<td>mvaddnstr()</td>
<td>addnstr()</td>
</tr>
<tr>
<td>mvaddstr()</td>
<td>addstr()</td>
</tr>
<tr>
<td>mvaddnwstr()</td>
<td>addnwstr()</td>
</tr>
<tr>
<td>mvaddwstr()</td>
<td>addwstr()</td>
</tr>
<tr>
<td>mvadd_wch()</td>
<td>add_wch()</td>
</tr>
<tr>
<td>mvadd_wchnstr()</td>
<td>add_wchnstr()</td>
</tr>
<tr>
<td>mvadd_wchstr()</td>
<td>add_wchstr()</td>
</tr>
<tr>
<td>mvchgnat()</td>
<td>chgat()</td>
</tr>
<tr>
<td>mvdelch()</td>
<td>delch()</td>
</tr>
<tr>
<td>mvgetch()</td>
<td>getch()</td>
</tr>
<tr>
<td>mvgetnstr()</td>
<td>getnstr()</td>
</tr>
<tr>
<td>mvgetstr()</td>
<td>getstr()</td>
</tr>
<tr>
<td>mvget_uestr()</td>
<td>get_uestr()</td>
</tr>
<tr>
<td>mvget_uwch()</td>
<td>get_uwch()</td>
</tr>
<tr>
<td>mvget_uwstr()</td>
<td>get_uwstr()</td>
</tr>
<tr>
<td>mvhline()</td>
<td>hline()</td>
</tr>
<tr>
<td>mvhline_set()</td>
<td>hline_set()</td>
</tr>
<tr>
<td>mvinch()</td>
<td>inch()</td>
</tr>
<tr>
<td>mvlnchnstr()</td>
<td>inchchnstr()</td>
</tr>
<tr>
<td>mvlnchstr()</td>
<td>inchchstr()</td>
</tr>
<tr>
<td>mvlnnstr()</td>
<td>innstr()</td>
</tr>
<tr>
<td>mvlnnstr()</td>
<td>innstr()</td>
</tr>
<tr>
<td>mvlns_nustr()</td>
<td>ins_nustr()</td>
</tr>
<tr>
<td>mvlns_uwch()</td>
<td>ins_uwch()</td>
</tr>
<tr>
<td>mvlns_uwstr()</td>
<td>ins_uwstr()</td>
</tr>
<tr>
<td>mvlnstr()</td>
<td>innstr()</td>
</tr>
<tr>
<td>mvlnwch()</td>
<td>in_wch()</td>
</tr>
<tr>
<td>mvlnwchstr()</td>
<td>in_wchstr()</td>
</tr>
<tr>
<td>mvlnwstr()</td>
<td>in_wstr()</td>
</tr>
<tr>
<td>mvln_wch()</td>
<td>in_wch()</td>
</tr>
<tr>
<td>mvln_wchnstr()</td>
<td>in_wchnstr()</td>
</tr>
<tr>
<td>mvln_wchstr()</td>
<td>in_wchstr()</td>
</tr>
<tr>
<td>mvcprintw()</td>
<td>amvcprintw()</td>
</tr>
<tr>
<td>mvcscanw()</td>
<td>mvcscanw()</td>
</tr>
<tr>
<td>mvchine()</td>
<td>mvcoline()</td>
</tr>
<tr>
<td>mvchine_set()</td>
<td>mvcoline_set()</td>
</tr>
</tbody>
</table>

### See Also

mvcur()

### Name

mvcur - output cursor movement commands to the terminal

### Synopsis

```c
#include <curses.h>

int mvcur(int oldrow, int oldcol, int newrow, int newcol);
```
Description
The mvcur() function outputs one or more commands to the terminal that move
the terminal's cursor to \((newrow, newcol)\), an absolute position on the terminal
screen. The \((oldrow, oldcol)\) arguments specify the former cursor position. Specifying
the former position is necessary on terminals that do not provide coordinate-based
movement commands. On terminals that provide these commands, Curses may
select a more efficient way to move the cursor based on the former position. If
\((newrow, newcol)\) is not a valid address for the terminal in use, mvcur() fails. If
\((oldrow, oldcol)\) is the same as \((newrow, newcol)\), then mvcur() succeeds without
taking any action. If mvcur() outputs a cursor movement command, it updates its
information concerning the location of the cursor on the terminal.

Return Value
Upon successful completion, mvcur() returns OK.
Otherwise, it returns ERR.

Errors
No errors are defined.

Application Usage
After use of mvcur(), the model Curses maintains of the state of the terminal might
not match the actual state of the terminal. The application should touch and
refresh the window before resuming conventional use of Curses.

See Also
doupdate(), is_linetouched(), <curses.h>.

mvderwin()

Name
mvderwin - define window coordinate transformation

Synopsis
#include <curses.h>

int mvderwin(WINDOW *win, int par_y, int par_x);

Description
The mvderwin() function specifies a mapping of characters. The function identifies
a mapped area of the parent of the specified window, whose size is the same as
the size of the specified window and whose origin is at \((par_y, par_x)\) of the parent
window.
• During any refresh of the specified window, the characters displayed in that
window’s display area of the terminal are taken from the mapped area.
• Any references to characters in the specified window obtain or modify
characters in the mapped area.

That is, mvderwin() defines a coordinate transformation from each position in the
mapped area to a corresponding position (same \(y, x\) offset from the origin) in the
specified window.

Chapter 5. Curses Interfaces  85
Return Value
Upon successful completion, mvderwin() returns OK. Otherwise, it returns ERR.

Errors
No errors are defined.

See Also
derwin(), doupdate(), dupwin(), <curses.h>.

mvprintw()

Name
mvprintw, mvwprintw, printw, wprintw - print formatted output in window

Synopsis
#include <curses.h>
int mvprintw(int y, int x, char *fmt, ...);
int mvwprintw(WINDOW *win, int y, int x, char *fmt, ...);
int printw(char *fmt, ...);
int wprintw(WINDOW *win, char *fmt, ...);

Description
The mvprintw(), mvwprintw(), printw() and wprintw() functions are analogous to printf(). The effect of these functions is as though sprintf() were used to format the string, and then waddstr() were used to add that multi-byte string to the current or specified window at the current or specified cursor position.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

See Also
addnstr(), fprintf(), <curses.h>

mvscanw()

Name
mvscanw, mvwscanw, scanw, wscanw - convert formatted input from a window

Synopsis
#include <curses.h>
int mvscanw(int y, int x, char *fmt, ...);
int mvwscanw(WINDOW *win, int y, int x, char *fmt, ...);
int scanw(char *fmt, ...);
int wscanw(WINDOW *win, char *fmt, ...);

Description
These functions are similar to scanf(). Their effect is as though mvwgetstr() were called to get a multi-byte character string from the current or specified window at the current or specified cursor position, and then sscanf() were used to interpret and convert that string.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

See Also
getnstr(), printw(), fscanf(), wcstombs(), <curses.h>.

mvwin()

Name
mvwin - move window

Synopsis
#include <curses.h>
int mvwin(WINDOW *win, int y, int x);

Description
The mvwin() function moves the specified window so that its origin is at position (y, x). If the move would cause any portion of the window to extend past any edge of the screen, the function fails and the window is not moved.

Return Value
Upon successful completion, mvwin() returns OK. Otherwise, it returns ERR.

Errors
No errors are defined.

Application Usage
The application should not move subwindows by calling mvwin().

See Also
derwin(), doupdate(), is_linetouched(), <curses.h>. 
napms()

Name
napms - suspend the calling process

Synopsis
#include <curses.h>
int napms(int ms);

Description
The napms() function takes at least ms milliseconds to return.

Return Value
The napms() function returns OK.

Errors
No errors are defined.

Application Usage
A more reliable method of achieving a timed delay is the usleep() function.

See Also
delay_output(), usleep() <curses.h>.

newpad()

Name
newpad, pnoutrefresh, prefresh, subpad - pad management functions

Synopsis
#include <curses.h>
WINDOW *newpad(int nlines, int ncols);

int pnoutrefresh(WINDOW *pad, int pminrow, int pmincol, int sminrow,
    int smincol, int smaxrow, int smaxcol);

int prefresh(WINDOW *pad, int pminrow, int pmincol, int sminrow,
    int smincol, int smaxrow, int smaxcol);

WINDOW *subpad(WINDOW *orig, int nlines, int ncols, int begin_y,
    int begin_x);

Description
The newpad() function creates a specialized WINDOW data structure representing
a pad with nlines lines and ncols columns. A pad is like a window, except that it is
not necessarily associated with a viewable part of the screen. Automatic refreshes
of pads do not occur.

The subpad() function creates a subwindow within a pad with nlines lines and
ncols columns. Unlike subwin(), which uses screen coordinates, the window is at
position \((\text{begin}_y, \text{begin}_x)\) on the pad. The window is made in the middle of the window \textit{orig}, so that changes made to one window affect both windows.

The \texttt{prefresh()} and \texttt{pnoutrefresh()} functions are analogous to \texttt{wrefresh()} and \texttt{wnoutrefresh()} except that they relate to pads instead of windows. The additional arguments indicate what part of the pad and screen are involved. The \texttt{pminrow} and \texttt{pmincol} arguments specify the origin of the rectangle to be displayed in the pad. The \texttt{sminrow}, \texttt{smincol}, \texttt{smaxrow} and \texttt{smaxcol} arguments specify the edges of the rectangle to be displayed on the screen. The lower right-hand corner of the rectangle to be displayed in the pad is calculated from the screen coordinates, since the rectangles must be the same size. Both rectangles must be entirely contained within their respective structures. Negative values of \texttt{pminrow}, \texttt{pmincol}, \texttt{sminrow} or \texttt{smincol} are treated as if they were zero.

Return Value

Upon successful completion, the \texttt{newpad()} and \texttt{subpad()} functions return a pointer to the pad data structure. Otherwise, they return a null pointer.

Upon successful completion, \texttt{pnoutrefresh()} and \texttt{prefresh()} return \texttt{OK}. Otherwise, they return \texttt{ERR}.

Errors

No errors are defined.

Application Usage

To refresh a pad, call \texttt{prefresh()} or \texttt{pnoutrefresh()}, not \texttt{wrefresh()}. When porting code to use pads from WINDOWS, remember that these functions require additional arguments to specify the part of the pad to be displayed and the location on the screen to be used for the display.

Although a subwindow and its parent pad may share memory representing characters in the pad, they need not share status information about what has changed in the pad. Therefore, after modifying a subwindow within a pad, it may be necessary to call \texttt{touchwin()} or \texttt{touchline()} on the pad before calling \texttt{prefresh()}.

See Also

d\texttt{erwin()}, \texttt{doupdate()}, \texttt{is\_linetouched()}, <\texttt{curses.h}>.

\texttt{newterm()}
newwin()

Name
newwin - create a new window

Synopsis
#include <curses.h>
WINDOW *newwin(int nlines, int ncols, int begin_y, int begin_x);

Description
Refer to derwin().

nl()

Name
nl, nonl - enable/disable newline translation

Synopsis
#include <curses.h>
int nl(void);
int nonl(void);

Description
The nl() function enables a mode in which carriage return is translated to newline on input. The nonl() function disables the above translation. Initially, the above translation is enabled.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

Application Usage
The default translation adapts the terminal to environments in which newline is the line termination character. However, by disabling the translation with nonl(), the application can sense the pressing of the carriage return key.

See Also
<curses.h>.

no

Name
no - pointer page for functions with no prefix
The no prefix indicates that a Curses function disables a mode. (The corresponding functions without the no prefix enable the same mode.)

The no functions are discussed together with the corresponding functions that do not have these prefixes. They are found on the following entries:

<table>
<thead>
<tr>
<th>Function</th>
<th>Refer to</th>
</tr>
</thead>
<tbody>
<tr>
<td>nocbreak()</td>
<td>cbreak()</td>
</tr>
<tr>
<td>noecho()</td>
<td>echo()</td>
</tr>
<tr>
<td>nonl()</td>
<td>nl()</td>
</tr>
<tr>
<td>noraw()</td>
<td>cbreak()</td>
</tr>
</tbody>
</table>

### nodelay()

**Name**

nodelay - enable or disable block during read

**Synopsis**

```c
#include <curses.h>

int nodelay(WINDOW *win, bool bf);
```

**Description**

The nodelay() function specifies whether Delay Mode or No Delay Mode is in effect for the screen associated with the specified window. If `bf` is TRUE, this screen is set to No Delay Mode. If `bf` is FALSE, this screen is set to Delay Mode. The initial state is FALSE.

**Return Value**

Upon successful completion, nodelay() returns OK. Otherwise, it returns ERR.

**Errors**

No errors are defined.

**See Also**

`getch()`, `halfdelay()`, `<curses.h>`.

### noqiflush()

**Name**

noqiflush, qiflush - enable/disable queue flushing

---

2. The nodelay() function has an entry under its own name because there is no corresponding delay() function. The noqiflush() and notimeout() functions have an entry under their own names because they precede the corresponding function without the no prefix in alphabetical order.
Synopsis

```c
#include <curses.h>

void noqiflush(void);

void qiflush(void);
```

Description

The qiflush() function causes all output in the display driver queue to be flushed whenever an interrupt key (interrupt, suspend, or quit) is pressed. The noqiflush() causes no such flushing to occur. The default for the option is inherited from the display driver settings.

Return Value

These functions do not return a value.

Errors

No errors are defined.

Application Usage

Calling qiflush() provides faster response to interrupts, but causes Curses to have the wrong idea of what is on the screen. The same effect is achieved outside Curses using the NOFLUSH local mode flag specified in the XBD specification (General Terminal Interface).

See Also

intrflush(), <curses.h>.

notimeout()  

Name

notimeout, timeout, wtimeout - control blocking on input

Synopsis

```c
#include <curses.h>

int notimeout(WINDOW *win, bool bf);

void timeout(int delay);

void wtimeout(WINDOW *win, int delay);
```

Description

The notimeout() function specifies whether Timeout Mode or No Timeout Mode is in effect for the screen associated with the specified window. If `bf` is TRUE, this screen is set to No Timeout Mode. If `bf` is FALSE, this screen is set to Timeout Mode. The initial state is FALSE.

The timeout() and wtimeout() functions set blocking or non-blocking read for the current or specified window based on the value of delay:

- `delay < 0`: One or more blocking reads (indefinite waits for input) are used.
delay = 0
One or more non-blocking reads are used. Any Curses input function will fail if every character of the requested string is not immediately available.

delay > 0
Any Curses input function blocks for delay milliseconds and fails if there is still no input.

Return Value
Upon successful completion, the notimeout() function returns OK. Otherwise, it returns ERR.

The timeout() and wtimeout() functions do not return a value.

Errors
No errors are defined.

See Also
getch(), halfdelay(), nodelay(), <curses.h>.

overlay()

Name
overlay, overwrite - copy overlapped windows

Synopsis
#include <curses.h>

int overlay(const WINDOW *srcwin, WINDOW *dstwin);
int overwrite(const WINDOW *srcwin, WINDOW *dstwin);

Description
The overlay() and overwrite() functions overlay srcwin on top of dstwin. The srcwin and dstwin arguments need not be the same size; only text where the two windows overlap is copied.

The overwrite() function copies characters as though a sequence of win_wch() and wadd_wch() were performed with the destination window’s attributes and background attributes cleared.

The overlay() function does the same thing, except that, whenever a character to be copied is the background character of the source window, overlay() does not copy the character but merely moves the destination cursor the width of the source background character.

If any portion of the overlaying window border is not the first column of a multi-column character then all the column positions will be replaced with the background character and rendition before the overlay is done. If the default background character is a multi-column character when this occurs, then these functions fail.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.
pair_content()

Name
pair_content, PAIR_NUMBER - get information on a color pair

Synopsis
#include <curses.h>

int pair_content(short pair, short *f, short *b);
int PAIR_NUMBER(int value);

Description
Refer to can_change_color().

pechochar()

Name
pechochar, pecho_wchar - write a character and rendition and immediately refresh the pad

Synopsis
#include <curses.h>

int pechochar(WINDOW *win, chtype ch);
int pecho_wchar(WINDOW *pad, const cchar_t *wch);

Description
The pechochar() and pecho_wchar() functions output one character to a pad and immediately refresh the pad. They are equivalent to a call to waddch() or wadd_wch(), respectively, followed by a call to refresh(). The last location of the pad on the screen is reused for the arguments to refresh().

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.
Application Usage

The pechochar() function is only guaranteed to operate reliably on character sets in which each character fits into a single byte, whose attributes can be expressed using only constants with the A_ prefix.

See Also

echochar(), echo_char(), newpad(), <curses.h>.

pnoutrefresh()

Name

pnoutrefresh, prefresh - refresh pads

Synopsis

#include <curses.h>

int pnoutrefresh(WINDOW *pad, int pminrow, int pmincol, int sminrow,
                 int smincol, int smaxrow, int smaxcol);

int prefresh(WINDOW *pad, int pminrow, int pmincol, int sminrow,
             int smincol, int smaxrow, int smaxcol);

Description

Refer to newpad().

printw()

Name

printw - print formatted output in the current window

Synopsis

#include <curses.h>

int printw(char *fmt, ...);

Description

Refer to mvprintw().

putp()

Name

putp, tputs - output commands to the terminal

Synopsis

#include <term.h>

int putp(const char *str);

int tputs(const char *str, int affcnt, int (*putfunc)(int));
Enhanced Curses

Description

These functions output commands contained in the terminfo database to the terminal.

The putp() function is equivalent to tputs(str, 1, putchar). The output of putp() always goes to stdout, not to the fildes specified in setupterm().

The tputs() function outputs str to the terminal. The str argument must be a terminfo string variable or the return value from tgetstr(), tgoto(), tigetstr() or tparm(). The affcnt argument is the number of lines affected, or 1 if not applicable. If the terminfo database indicates that the terminal in use requires padding after any command in the generated string, tputs() inserts pad characters into the string that is sent to the terminal, at positions indicated by the terminfo database. The tputs() function outputs each character of the generated string by calling the user-supplied function putfunc (see below).

The user-supplied function putfunc (specified as an argument to tputs()) is either putchar() or some other function with the same prototype. The tputs() function ignores the return value of putfunc.

Return Value

Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors

No errors are defined.

Application Usage

After use of any of these functions, the model Curses maintains of the state of the terminal might not match the actual state of the terminal. The application should touch and refresh the window before resuming conventional use of Curses.

Use of these functions requires that the application contain so much information about a particular class of terminal that it defeats the purpose of using Curses.

On some terminals, a command to change rendition conceptually occupies space in the screen buffer (with or without width). Thus, a command to set the terminal to a new rendition would change the rendition of some characters already displayed.

See Also
doupdate(), is_linetouched(), putchar(), tgetent(), tigetflag(), <term.h>.

putwin()

Name

putwin - dump window to a file

Synopsis

#include <curses.h>

int putwin(WINDOW *win, FILE *filep);


**Description**

Refer to getwin().

---

### qiflush()

**Name**

qiflush - enable queue flushing

**Synopsis**

```c
#include <curses.h>

void qiflush(void);
```

**Description**

Refer to noqiflush().

---

### raw()

**Name**

raw - set Raw Mode

**Synopsis**

```c
#include <curses.h>

int raw(void);
```

**Description**

Refer to cbreak().

---

### redrawwin()

**Name**

redrawwin, wredrawln - line update status functions

**Synopsis**

```c
#include <curses.h>

int redrawwin(WINDOW *win);

int wredrawln(WINDOW *win, int beg_line, int num_lines);
```

**Description**

The redrawwin() and wredrawln() functions inform the implementation that some or all of the information physically displayed for the specified window may have been corrupted. The redrawwin() function marks the entire window; wredrawln() marks only num_lines lines starting at line number beg_line. The functions prevent the next refresh operation on that window from performing any optimization based on assumptions about what is physically displayed there.
Enhanced Curses

Return Value
Upon successful completion, these functions return OK. Otherwise they return ERR.

Errors
No errors are defined.

Application Usage
The redrawwin() and wredrawln() functions could be used in a text editor to implement a command that redraws some or all of the screen.

See Also
clearok(), doupdate(), <curses.h>.

refresh()

Name
refresh - refresh current window

Synopsis
#include <curses.h>
int refresh(void);

Description
Refer to doupdate().

reset_prog_mode()

Name
reset_prog_mode, reset_shell_mode - restore program or shell terminal modes

Synopsis
#include <curses.h>
int reset_prog_mode(void);
int reset_shell_mode(void);

Description
Refer to def_prog_mode().

resetty()

Name
resetty, savetty - save/restore terminal mode
Synopsis
#include <curses.h>
int resetty(void);
int savetty(void);

Description
The resetty() function restores the program mode as of the most recent call to savetty().

The savetty() function saves the state that would be put in place by a call to reset_prog_mode().

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

See Also
def_prog_mode(), <curses.h>.

restartterm()

Name
restartterm - change terminal type

Synopsis
#include <term.h>
int restartterm(char *term, int fildes, int *errret);

Description
Refer to del_curterm().

riпoffline()

Name
рипoffline - reserve a line for a dedicated purpose

Synopsis
#include <curses.h>
int ripoffline(int line, int (*init)(WINDOW *win, int columns));

Description
The ripoffline() function reserves a screen line for use by the application.
Enhanced Curses

Any call to ripoffline() must precede the call to initscr() or newterm(). If line is positive, one line is removed from the beginning of stdscr; if line is negative, one line is removed from the end. Removal occurs during the subsequent call to initscr() or newterm(). When the subsequent call is made, the function pointed to by init is called with two arguments: a WINDOW pointer to the one-line window that has been allocated and an integer with the number of columns in the window. The initialization function cannot use the LINES and COLS external variables and cannot call wrefresh() or doupdate(), but may call wnoutrefresh().

Up to five lines can be ripped off. Calls to ripoffline() above this limit have no effect but report success.

Return Value

The ripoffline() function returns OK.

Errors

No errors are defined.

Application Usage

Calling slk_init() reduces the size of the screen by one line if initscr() eventually uses a line from stdscr to emulate the soft labels. If slk_init() rips off a line, it thereby reduces by one the number of lines an application can reserve by subsequent calls to ripoffline(). Thus, portable applications that use soft label functions should not call ripoffline() more than four times.

When initscr() or newterm() calls the initialization function pointed to by init, the implementation may pass NULL for the WINDOW pointer argument win. This indicates inability to allocate a one-line window for the line that the call to ripoffline() ripped off. Portable applications should verify that win is not NULL before performing any operation on the window it represents.

See Also

doupdate(), initscr(), slk_atroff(), <curses.h>.

savetty()

Name

savetty - save terminal mode

Synopsis

#include <curses.h>

int savetty(void);

Description

Refer to resetty().

scanw()

Name

scanw - convert formatted input from the current window
Synopsis
#include <curses.h>
int scanw(char *fmt, ...);

Description
Refer to mvscanw().

scr_dump()

Name
scr_dump, scr_init, scr_restore, scr_set - screen file input/output functions

Synopsis
#include <curses.h>
int scr_dump(const char *filename);
int scr_init(const char *filename);
int scr_restore(const char *filename);
int scr_set(const char *filename);

Description
The scr_dump() function writes the current contents of the virtual screen to the file named by filename in an unspecified format.

The scr_restore() function sets the virtual screen to the contents of the file named by filename, which must have been written using scr_dump(). The next refresh operation restores the screen to the way it looked in the dump file.

The scr_init() function reads the contents of the file named by filename and uses them to initialize the Curses data structures to what the terminal currently has on its screen. The next refresh operation bases any updates on this information, unless either of the following conditions is true:
• The terminal has been written to since the virtual screen was dumped to filename
• The terminfo capabilities rmcup and nrrmc are defined for the current terminal.

The scr_set() function is a combination of scr_restore() and scr_init(). It tells the program that the information in the file named by filename is what is currently on the screen, and also what the program wants on the screen. This can be thought of as a screen inheritance function.

Return Value
On successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.
Application Usage

The scr_init() function is called after initscr() or a system() call to share the screen with another process that has done a scr_dump() after its endwin() call.

To read a window from a file, call getwin(); to write a window to a file, call putwin().

See Also

delscreen(), doupdate(), endwin(), getwin(), open(), read(), write(), <curses.h>

scrl()

Name

scrl, scroll, wscrl - scroll a Curses window

Synopsis

#include <curses.h>

int scrl(int n);

int scroll(WINDOW *win);

int wscrl(WINDOW *win, int n);

Description

The scroll() function scrolls win one line in the direction of the first line.

The scrl() and wscrl() functions scroll the current or specified window. If n is positive, the window scrolls n lines toward the first line. Otherwise, the window scrolls -n lines toward the last line.

These functions do not change the cursor position. If scrolling is disabled for the current or specified window, these functions have no effect.

Return Value

Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors

No errors are defined.

See Also

<curses.h>.

scrollok()

Name

scrollok - enable or disable scrolling on a window
Synopsis
#include <curses.h>

int scrollok(WINDOW *win, bool bf);

Description
Refer to clearok().

setcchar()

Name
setcchar - set cchar_t from a wide character string and rendition

Synopsis
#include <curses.h>

int setcchar(cchar_t *wcval, const wchar_t *wch, const attr_t attrs,
short color_pair, const void *opts);

Description
The setcchar() function initializes the object pointed to by wcval according to the character attributes in attrs, the color pair in color_pair and the wide character string pointed to by wch.

The opts argument is reserved for definition in a future edition of this document. Currently, the application must provide a null pointer as opts.

Return Value
Upon successful completion, setcchar() returns OK. Otherwise, it returns ERR.

Errors
No errors are defined.

See Also
attroff(), can_change_color(), getcchar(), <curses.h>.

set_curterm()

Name
set_curterm - set current terminal

Synopsis
#include <term.h>

TERMINAL *set_curterm(TERMINAL *nterm);

Description
Refer to del_curterm().
setscrreg()

Name

setscrreg, wsetscrreg - define software scrolling region

Synopsis

```
#include <curses.h>

int setscrreg(int top, int bot);
int wsetscrreg(WINDOW *win, int top, int bot);
```

Description

Refer to clearok().

setupterm()

Name

setupterm - access the terminfo database
Synopsis
#include <term.h>

int setupterm(char *term, int fildes, int *errret);

Description
Refer to del_curterm().

slk_attroff()

Name

slk_attroff, slk_attr_off, slk_attron, slk_attr_on, slk_attrset, slk_attr_set, slk_clear,
slk_color, slk_init, slk_label, slk_noutrefresh, slk_refresh, slk_restore, slk_set,
slk_touch, slk_wset - soft label functions

Synopsis
#include <curses.h>

int slk_attroff(const chtype attrs);
int slk_attr_off(const attr_t attrs, void *opts);
int slk_attron(const chtype attrs);
int slk_attr_on(const attr_t attrs, void *opts);
int slk_attrset(const chtype attrs);
int slk_attr_set(const attr_t attrs, short color_pair_number, void *opts);
int slk_clear(void);
in slk_color(short color_pair_number);
int slk_init(int fmt);
char *slk_label(int labnum);
int slk_noutrefresh(void);
int slk_refresh(void);
int slk_restore(void);
int slk_set(int labnum, const char *label, int justify);
int slk_touch(void);
in slk_wset(int labnum, const wchar_t *label, int justify);

Description
The Curses interface manipulates the set of soft function-key labels that exist on many terminals. For those terminals that do not have soft labels, Curses takes over the bottom line of stdscr, reducing the size of stdscr and the value of the LINES external variable. There can be up to eight labels of up to eight display columns each.
To use soft labels, `slk_init()` must be called before `initscr()`, `newterm()` or `ripoffline()` is called. If `initscr()` eventually uses a line from `stdscr` to emulate the soft labels, then `fmt` determines how the labels are arranged on the screen. Setting `fmt` to 0 indicates a 3-2-3 arrangement of the labels; 1 indicates a 4-4 arrangement. Other values for `fmt` are unspecified.

The `slk_init()` function has the effect of calling `ripoffline()` to reserve one screen line to accommodate the requested format.

The `slk_set()` and `slk_wset()` functions specify the text of soft label number `labnum`, within the range from 1 to and including 8. The `label` argument is the string to be put on the label. With `slk_set()`, and `slk_wset()`, the width of the label is limited to eight column positions. A null string or a null pointer specifies a blank label. The `justify` argument can have the following values to indicate how to justify `label` within the space reserved for it:

- 0: Align the start of label with the start of the space
- 1: Center label within the space
- 2: Align the end of label with the end of the space

The `slk_refresh()` and `slk_noutrefresh()` functions correspond to the `wrefresh()` and `wnoutrefresh()` functions.

The `slk_label()` function obtains soft label number `labnum`.

The `slk_clear()` function immediately clears the soft labels from the screen.

The `slk_restore()` function immediately restores the soft labels to the screen after a call to `slk_clear()`.

The `slk_touch()` function forces all the soft labels to be output the next time `slk_noutrefresh()` or `slk_refresh()` is called.

The `slk_attron()`, `slk_attroff()`, `slk_attrset()` and `slk_color()` functions correspond to `attron()`, `attroff()`, `attrset()`, and `color_set()`. They have an effect only if soft labels are simulated on the bottom line of the screen.

The `slk_attr_off()`, `slk_attr_on()`, `slk_attr_set()`, and `slk_color()` functions correspond to `slk_attroff()`, `slk_attron()`, `slk_attrset()` and `color_set()` and thus support the attribute constants with WA_prefix and color.

The `opts` argument is reserved for definiton in a future edition of this document. Currently, the application must provide a null pointer as `opts`.

**Return Value**

Upon successful completion, `slk_label()` returns the requested label with leading and trailing blanks stripped. Otherwise, it returns a null pointer.

Upon successful completion, the other functions return OK. Otherwise, they return ERR.

**Errors**

No errors are defined.
Application Usage

When using multi-byte character sets, applications should check the width of the string by calling `mbstowcs()` and then `wcswidth()` before calling `slk_set()`. When using wide characters, applications should check the width of the string by calling `wcswidth()` before calling `slk_set()`.

Since the number of columns that a wide character string will occupy is codeset-specific, call `wcwidth()` and `wcswidth()` to check the number of column positions in the string before calling `slk_wset()`.

Most applications would use `slk_noutrefresh()` because a `wrefresh()` is likely to follow soon.

See Also

`attr_get()`, `attroff()`, `delscreen()`, `mbstowcs()`, `ripoffline()`, `wcswidth()`, `<curses.h>`.

standend()

Name

standend, standout, wstandend, wstandout - set and clear window attributes

Synopsis

```c
#include <curses.h>
int standend(void);
int standout(void);
int wstandend(WINDOW *win);
int wstandout(WINDOW *win);
```

Description

The `standend()` and `wstandend()` functions turn off all attributes of the current or specified window.

The `standout()` and `wstandout()` functions turn on the standout attribute of the current or specified window.

Return Value

These functions always return 1.

Errors

No errors are defined.

See Also

`attroff()`, `attr_get()`, `<curses.h>`.
start_color()

Name
start_color - initialize use of colors on terminal

Synopsis
#include <curses.h>
int start_color(void);

Description
Refer to can_change_color().

stdscr

Name
stdscr - default window

Synopsis
#include <curses.h>
extern WINDOW *stdscr;

Description
The external variable stdscr specifies the default window used by functions that do not specify a window using an argument of type WINDOW *. Other windows may be created using newwin().

See Also
derwin(), <curses.h>.

subpad()

Name
subpad - create a subwindow in a pad

Synopsis
#include <curses.h>
WINDOW *subpad(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);

Description
Refer to newpad().

subwin()

Name
subwin - create a subwindow
Synopsis

```c
#include <curses.h>

WINDOW *subwin(WINDOW *orig, int nlines, int ncols, int begin_y, int begin_x);
```

Description

Refer to derwin().

**syncok()**

Name

cyncok, wcursyncup, wsyncdown, wsyncup - synchronise a window with its parents or children

Synopsis

```c
#include <curses.h>

int syncok(WINDOW *win, bool bf);
void wcursyncup(WINDOW *win);
void wsyncdown(WINDOW *win);
void wsyncup(WINDOW *win);
```

Description

The syncok() function determines whether all ancestors of the specified window are implicitly touched whenever there is a change in the window. If \( \text{bf} \) is TRUE, such implicit touching occurs. If \( \text{bf} \) is FALSE, such implicit touching does not occur. The initial state is FALSE.

The wcursyncup() function updates the current cursor position of the ancestors of \( \text{win} \) to reflect the current cursor position of \( \text{win} \).

The wsyncdown() function touches \( \text{win} \) if any ancestor window has been touched.

The wsyncup() function unconditionally touches all ancestors of \( \text{win} \).

Return Value

Upon successful completion, syncok() returns OK. Otherwise, it returns ERR.

The other functions do not return a value.

Errors

No errors are defined.

Application Usage

Applications seldom call wsyncdown() because it is called by all refresh operations.

See Also

doupdate(), is_linetouched(), <curses.h>.
termattr

Name
termattr - get supported terminal video attributes

Synopsis
#include <curses.h>

cttype termattr(void);
attr_t term_attr(void);

Description
The termattr() function extracts the video attributes of the current terminal which is supported by the chtype data type.

The term_attr() function extracts information for the video attributes of the current terminal which is supported for a ccha_t.

Return Value
The termattr() function returns a logical OR of A_values of all video attributes supported by the terminal. The term_attr() function returns a logical OR of WA_values of all video attributes supported by the terminal.

Errors
No errors are defined.

See Also
attroff(), attr_get(), <curses.h>.

tername

Name
tername - get terminal name

Synopsis
#include <curses.h>

char *tername(void);

Description
The tername() function obtains the terminal name as recorded by setupterm().

Return Value
The tername() function returns a pointer to the terminal name.

Errors
No errors are defined.
See Also
del_curterm(), getenv() initscr(), <curses.h>.

tgetent() 

Name
tgetent, tgetflag, tgetnum, tgetstr, tgoto - termcap database emulation (TO BE WITHDRAWN)

Synopsis
#include <term.h>

int tgetent(char *bp, const char *name);
int tgetflag(char id[2]);
int tgetnum(char id[2]);
char *tgetstr(char id[2], char **area);
char *tgoto(char *cap, int col, int row);

Description
The tgetent() function looks up the termcap entry for name. The emulation ignores the buffer pointer bp.

The tgetflag() function gets the boolean entry for id.

The tgetnum() function gets the numeric entry for id.

The tgetstr() function gets the string entry for id. If area is not a null pointer and does not point to a null pointer, tgetstr() copies the string entry into the buffer pointed to by *area and advances the variable pointed to by area to the first byte after the copy of the string entry.

The tgoto() function instantiates the parameters col and row into capability cap and returns a pointer to the resulting string.

All of the information available in the terminfo database need not be available through these functions.

Return Value
Upon successful completion, functions that return an integer return OK. Otherwise, they return ERR.

Functions that return pointers return a null pointer on error.

Errors
No errors are defined.
Application Usage

These functions are included as a conversion aid for programs that use the termcap library. Their arguments are the same and the functions are emulated using the terminfo database.

These functions are only guaranteed to operate reliably on character sets in which each character fits into a single byte, whose attributes can be expressed using only constants with the A_ prefix.

Any terminal capabilities from the terminfo database that cannot be retrieved using these interfaces can be retrieved using the interfaces described on the tigetflag() page.

Portable applications must use tputs() to output the strings returned by tgetstr() and tgoto().

See Also

putc(), setupterm(), tigetflag(), <term.h>.

tigetflag()

Name
tigetflag, tigetnum, tigetstr, tparm - retrieve capabilities from the terminfo database

Synopsis

#include <term.h>

int tigetflag(char *capname);

int tigetnum(char *capname);

char *tigetstr(char *capname);

char *tparm(char *cap, long p1, long p2, long p3, long p4,
        long p5, long p6, long p7, long p8, long p9);

Description

The tigetflag(), tigetnum(), and tigetstr() functions obtain boolean, numeric and string capabilities, respectively, from the selected record of the terminfo database. For each capability, the value to use as capname appears in the Capname column.

The tparm() function takes as cap a string capability. If cap is parameterized, tparm() resolves the parameterization. If the parameterized string refers to parameters %p1 through %p9, then tparm() substitutes the values of p1 through p9, respectively.

Return Value

Upon successful completion, tigetflag(), tigetnum() and tigetstr() return the specified capability. The tigetflag() function returns -1 if capname is not a boolean capability. The tigetnum() function returns -2 if capname is not a numeric capability. The tigetstr() function returns (char *)-1 if capname is not a string capability.

Upon successful completion, tparm() returns str with parameterization resolved. Otherwise, it returns a null pointer.
Errors

No errors are defined.

Application Usage

For parameterized string capabilities, the application should pass the return value from tigetstr() to tparm(), as described above.

Applications intending to send terminal capabilities directly to the terminal (which should only be done using tputs() or putp()) instead of using Curses, normally should obey the following rules:

- Call reset_shell_mode() to restore the display modes before exiting.
- If using cursor addressing, output enter_ca_mode upon startup and output exit_ca_mode before exiting.
- If using shell escapes, output exit_ca_mode and call reset_shell_mode() before calling the shell; call reset_prog_mode() and output enter_ca_mode after returning from the shell.

All parameterized terminal capabilities defined in this document can be passed to tparm(). Some implementations create their own capabilities, create capabilities for non-terminal devices, and redefine the capabilities in this document. These practices are non-conforming because it may be that tparm() cannot parse these user-defined strings.

See Also

def_prog_mode(), tgetent(), putp(), <term.h>.

timeout()

Name

timeout - control blocking on input

Synopsis

#include <curses.h>

void timeout(int delay);

Description

Refer to notimeout().

touchline()

Name

touchline, touchwin - window refresh control functions

Synopsis

#include <curses.h>

int touchline(WINDOW *win, int start, int count);

int touchwin(WINDOW *win);
Curses

Description
Refer to is_linetouched().

tparm()

Name
tparm - retrieve capabilities from the terminfo database

Synopsis
#include <term.h>
char *tparm(char *cap, long p1, long p2, long p3, long p4,
long p5, long p6, long p7, long p8, long p9);

Description
Refer to tigetflag().

tputs()

Name
tputs - output commands to the terminal

Synopsis
#include <curses.h>
int tputs(const char *str, int affcnt, int (*putfunc)(int));

Description
Refer to putp().
typeahead()

Name
typeahead - control checking for typeahead

Synopsis
#include <curses.h>
int typeahead(int fildes);

Description
The typeahead() function controls the detection of typeahead during a refresh,
based on the value of fildes:
• If fildes is a valid file descriptor, typeahead is enabled during refresh; Curses
  periodically checks fildes for input and aborts the refresh if any character is
  available. (This is the initial setting, and the typeahead file descriptor
  corresponds to the input file associated with the screen created by initscr() or
  newterm().) The value of fildes need not be the file descriptor on which the
  refresh is occurring.
• If fildes is -1, Curses does not check for typeahead during refresh.
Return Value

Upon successful completion, typeahead() returns OK. Otherwise, it returns ERR.

Errors

No errors are defined.

See Also
doupdate(), getch(), initscr(), <curses.h>.

unctrl()

Name
unctrl - generate printable representation of a character

Synopsis

#include <unctrl.h>

c char *unctrl(chtype c);

Description

The unctrl() function generates a character string that is a printable representation of c. If c is a control character, it is converted to the ^X notation. If c contains rendition information, the effect is undefined.

Return Value

Upon successful completion, unctrl() returns the generated string. Otherwise, it returns a null pointer.

Errors

No errors are defined.

See Also

keyname(), wunctrl(), <unctrl.h>.

ungetch()

Name
ungetch, unget_wch - push a character onto the input queue

Synopsis

#include <curses.h>

int ungetch(int ch);

int unget_wch(const wchar_t wch);

Description

The ungetch() function pushes the single-byte character ch onto the head of the input queue.
Enhanced Curses

The unget_wch() function pushes the wide character wch onto the head of the input queue.

One character of push-back is guaranteed. If these functions are called too many times without an intervening call to getch() or get_wch(), the operation may fail.

Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

See Also
getch(), get_wch(), <curses.h>.

untouchwin()

Name
untouchwin - window refresh control function

Synopsis
#include <curses.h>
int untouchwin(WINDOW *win);

Description
Refer to is_linetouched().

use_env()

Name
use_env - specify source of screen size information

Synopsis
#include <curses.h>
void use_env(bool boolval);

Description
The use_env() function specifies the technique by which the implementation determines the size of the screen. If boolval is FALSE, the implementation uses the values of lines and columns specified in the terminfo database. If boolval is TRUE, the implementation uses the LINES and COLUMNS environment variables. The initial value is TRUE.

Any call to use_env() must precede calls to initscr(), newterm() or setupterm().

Return Value
The function does not return a value.
Errors

No errors are defined.

See Also

del_curterm(), initscr(), <curses.h>.

vidattr()

Name

vidattr, vid_attr, vidputs, vid_puts - output attributes to the terminal

Synopsis

#include <curses.h>

int vidattr(chtype attr);

int vid_attr(attr_t attr, short color_pair_number, void *opt);

int vidputs(chtype attr, int (*putfunc)(int));

int vid_puts(attr_t attr, short_pair_number, void *opt, int_t (*putfunc)(init_t));

Description

These functions output commands to the terminal that change the terminal’s attributes.

If the terminfo database indicates that the terminal in use can display characters in the rendition specified by attr, then vidattr() outputs one or more commands to request that the terminal display subsequent characters in that rendition. The function outputs by calling putchar(). The vidattr() function neither relies on nor updates the model that Curses maintains of the prior rendition mode.

The vidputs() function computes the same terminal output string that vidattr() does, based on attr, but vidputs() outputs by calling the user-supplied function putfunc. The vid_attr() and vid_puts() functions correspond to vidattr() and vidputs() respectively, but take a set of arguments, one of type attr_t for the attributes, short for the color_pair_number and a void* and thus support the attribute constants with the WA_prefix.

The opts argument is reserved for definition in a future edition of this document. Currently, the application must provide a null pointer as opts.

The user-supplied function putfunc (specified as an argument to vidputs()) is either putchar() or some other function with the same prototype. The vidputs() function ignores the return value of putfunc.

The vid_attr() and vid_puts() functions correspond to vidattr() and vidputs(), respectively, but take an argument of type attr_t and thus support the attribute constants with the WA_prefix.

The user-supplied function putwfunc (specified as an argument to vid_puts()) is either putwchar() or some other function with the same prototype. The vid_puts() function ignores the return value of putwfunc.
Return Value
Upon successful completion, these functions return OK. Otherwise, they return ERR.

Errors
No errors are defined.

Application Usage
After use of any of these functions, the model Curses maintains of the state of the terminal might not match the actual state of the terminal. The application should touch and refresh the window before resuming conventional use of Curses.

Use of these functions requires that the application contain so much information about a particular class of terminal that it defeats the purpose of using Curses.

On some terminals, a command to change rendition conceptually occupies space in the screen buffer (with or without width). Thus, a command to set the terminal to a new rendition would change the rendition of some characters already displayed.

See Also
doupdate(), is_linetouched(), putchar(), putwchar(), tigetflag(), <curses.h>.

vline()

Name
vline - draw vertical line

Synopsis
#include <curses.h>
int vline(chtype ch, int n);

Description
Refer to hline().

vline_set()

Name
vline_set - draw vertical line from complex character and rendition

Synopsis
#include <curses.h>
int vline_set(const cchar_t *ch, int n);

Description
Refer to hline_set().
vwprintw()

Name
vwprintw - print formatted output in window

Synopsis
#include <varargs.h>
#include <curses.h>

int vwprintw(WINDOW *, char *, va_list varglist);

Description
The vwprintw() function achieves the same effect as wprintw() using a variable
argument list. The third argument is a va_list, as defined in <varargs.h>.

Return Value
Upon successful completion, vwprintw() returns OK. Otherwise, it returns ERR.

Errors
No errors are defined.

Application Usage
The vwprintw() function is deprecated because it relies on deprecated functions in
the XSH specification. The vw_printw() function is preferred. The use of the
vwprintw() and the vw_printw() functions in the same file will not work, due to
the requirements to include varargs.h and stdarg.h which both contain definitions
of va_list.

See Also
mvprintw(), fprintf(), vw_printw(), <curses.h>, <varargs.h>.

vw_printw()

Name
vw_printw - print formatted output in window

Synopsis
#include <stdarg.h>
#include <curses.h>

int vw_printw(WINDOW *, char *, va_list varglist);

Description
The vw_printw() function achieves the same effect as wprintw() using a variable
argument list. The third argument is a va_list, as defined in <stdarg.h>.

Return Value
Upon successful completion, vw_printw() returns OK. Otherwise, it returns ERR.
Errors
No errors are defined.

Application Usage
The vw_printw() function is preferred over vwprintw(). The use of the vwprintw() and the vw_printw() functions in the same file will not work, due to the requirement to include varargs.h and stdarg.h which both contain definitions of va_list.

See Also
mvprintw(), fprintf(), <curses.h>, <stdarg.h>.

vwscanw()

Name
vwscanw - convert formatted input from a window

Synopsis
#include <varargs.h>
#include <curses.h>

int vwscanw(WINDOW *, char *, va_list varglist);

Description
The vwscanw() function achieves the same effect as wscanw() using a variable argument list. The third argument is a va_list, as defined in <varargs.h>.

Return Value
Upon successful completion, vwscanw() returns OK. Otherwise, it returns ERR.

Errors
No errors are defined.

Application Usage
The vwscanw() function is deprecated because it relies on deprecated functions in the XSH specification. The vw_scanw() function is preferred. The use of the vwscanw() and the vw_scanw() functions in the same file will not work, due to the requirement to include varargs.h and stdarg.h which both contain definitions of va_list.

See Also
fscanf(), mvscanw(), vw_scanw(), <curses.h>, varargs.h>.

vw_scanw()

Name
vw_scanw - convert formatted input from a window
Synopsis

```
#include <stdarg.h>
#include <curses.h>

int vw_scanw(WINDOW *, char *, va_list varglist);
```

Description

The `vw_scanw()` function achieves the same effect as `wscanw()` using a variable argument list. The third argument is a `va_list`, as defined in `<stdarg.h>`.

Return Value

Upon successful completion, `vw_scanw()` returns OK. Otherwise, it returns ERR.

Errors

No errors are defined.

Application Usage

The `vw_scanw()` function is preferred over `vwscanw()`. The use of the `vwscanw()` and the `vw_scanw()` functions in the same file will not work, due to the requirement to include `vargs.h` and `stdarg.h` which both contain definitions of `va_list`.

See Also

`fscanf()`, `mvscanw()`, `<curses.h>`, `<stdarg.h>`.

---

**W**

Name

w - pointer page for functions with w prefix

Description

Most uses of the w prefix indicate that a Curses function takes a win argument that specifies the affected window.³ (The corresponding functions without the w prefix operate on the current window.)

The w functions are discussed together with the corresponding functions without the w prefix. They are found on the following entries:

<table>
<thead>
<tr>
<th>Function</th>
<th>Refer to</th>
</tr>
</thead>
<tbody>
<tr>
<td>* There is no corresponding function without the w prefix.</td>
<td></td>
</tr>
<tr>
<td><code>waddch()</code></td>
<td><code>addch()</code></td>
</tr>
<tr>
<td><code>waddchstr()</code></td>
<td><code>addchstr()</code></td>
</tr>
<tr>
<td><code>waddstr()</code></td>
<td><code>addnstr()</code></td>
</tr>
<tr>
<td><code>waddnwstr()</code></td>
<td><code>addnwstr()</code></td>
</tr>
<tr>
<td><code>wadd_wch()</code></td>
<td><code>add_wch()</code></td>
</tr>
<tr>
<td><code>wadd_wchstr()</code></td>
<td><code>add_wchstr()</code></td>
</tr>
</tbody>
</table>

³. The wunctrl() function is an exception to this rule and has an entry under its own name.
### Curses

<table>
<thead>
<tr>
<th>Function</th>
<th>Refer to</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wadd_wchstr()</code></td>
<td></td>
</tr>
<tr>
<td><code>wattroff()</code></td>
<td></td>
</tr>
<tr>
<td><code>wattron()</code></td>
<td></td>
</tr>
<tr>
<td><code>wattr_set()</code></td>
<td></td>
</tr>
<tr>
<td><code>wbkgd()</code></td>
<td></td>
</tr>
<tr>
<td><code>wbkgdset()</code></td>
<td></td>
</tr>
<tr>
<td><code>wbkgrnd()</code></td>
<td></td>
</tr>
<tr>
<td><code>wbkgrndset()</code></td>
<td></td>
</tr>
<tr>
<td><code>wborder()</code></td>
<td></td>
</tr>
<tr>
<td><code>wborder_set()</code></td>
<td></td>
</tr>
<tr>
<td><code>wchga()t</code></td>
<td></td>
</tr>
<tr>
<td><code>wclear()</code></td>
<td></td>
</tr>
<tr>
<td><code>wclrtoeol()</code></td>
<td></td>
</tr>
<tr>
<td><code>wcsyncup()</code></td>
<td></td>
</tr>
<tr>
<td><code>wdech()</code></td>
<td></td>
</tr>
<tr>
<td><code>wdeleteln()</code></td>
<td></td>
</tr>
<tr>
<td><code>wechochar()</code></td>
<td></td>
</tr>
<tr>
<td><code>wecho_wchar()</code></td>
<td></td>
</tr>
<tr>
<td><code>werase()</code></td>
<td></td>
</tr>
<tr>
<td><code>wgetbkgrnd()</code></td>
<td></td>
</tr>
<tr>
<td><code>wget()ch</code></td>
<td></td>
</tr>
<tr>
<td><code>wgetnstr()</code></td>
<td></td>
</tr>
<tr>
<td><code>wgetn_wstr()</code></td>
<td></td>
</tr>
<tr>
<td><code>wgetstr()</code></td>
<td></td>
</tr>
<tr>
<td><code>wget_wch()</code></td>
<td></td>
</tr>
<tr>
<td><code>wget_wstr()</code></td>
<td></td>
</tr>
<tr>
<td><code>whline()</code></td>
<td></td>
</tr>
<tr>
<td><code>whline_set()</code></td>
<td></td>
</tr>
<tr>
<td><code>winch()</code></td>
<td></td>
</tr>
<tr>
<td><code>winchnstr()</code></td>
<td></td>
</tr>
<tr>
<td><code>winchstr()</code></td>
<td></td>
</tr>
<tr>
<td><code>winnstr()</code></td>
<td></td>
</tr>
<tr>
<td><code>winnwstr()</code></td>
<td></td>
</tr>
<tr>
<td><code>winsch()</code></td>
<td></td>
</tr>
<tr>
<td><code>winsdelln()</code></td>
<td></td>
</tr>
<tr>
<td><code>winsertln()</code></td>
<td></td>
</tr>
<tr>
<td><code>winsnstr()</code></td>
<td></td>
</tr>
<tr>
<td><code>winsstr()</code></td>
<td></td>
</tr>
<tr>
<td><code>winswstr()</code></td>
<td></td>
</tr>
<tr>
<td><code>wins_wch()</code></td>
<td></td>
</tr>
<tr>
<td><code>wins_wstr()</code></td>
<td></td>
</tr>
<tr>
<td><code>win_wch()</code></td>
<td></td>
</tr>
<tr>
<td><code>win_wchnstr()</code></td>
<td></td>
</tr>
<tr>
<td><code>win_wchstr()</code></td>
<td></td>
</tr>
<tr>
<td><code>wmovew()</code></td>
<td></td>
</tr>
<tr>
<td><code>wnoutrefresh()</code></td>
<td></td>
</tr>
<tr>
<td><code>wprintw()</code></td>
<td></td>
</tr>
</tbody>
</table>

---

122  z/OS V2R1.0 C Curses
Function | Refer to
--- | ---
`wredrawln()` | `redrawln()`
`wrefresh()` | `doupdate()`
`wscanw()` | `mvscanw()`
`wrepl()` | `scrl()`
`wsetscreg()` | `clearok()`
`wstandend()` | `standend()`
`wstandout()` | `standend()`
`wsysdown()` | `syncok()`
`wtruncup()` | `syncok()`
`wttimeout()` | `notimeout()`
`wtimeout()` | `is_linetouch()`
`wunctrlln()` | `hline()`
`wvline()` | `hline_set()`

---

**wunctrl()**

**Name**

`wunctrl` - generate printable representation of a wide character

**Synopsis**

```c
#include <curses.h>

wchar_t *wunctrl(cchar_t *wc);
```

**Description**

The `wunctrl()` function generates a wide character string that is a printable representation of the wide character `wc`.

This function also performs the following processing on the input argument:

- Control characters are converted to the `^X` notation.
- Any rendition information is removed.

**Return Value**

Upon successful completion, `wunctrl()` returns the generated string. Otherwise, it returns a null pointer.

**Errors**

No errors are defined.

**See Also**

`keyname()`, `unctrl()`, `<curses.h>`.
Enhanced Curses
Chapter 6. Headers

This chapter describes the contents of headers used by the Curses functions, macros and external variables.

Headers contain the definition of symbolic constants, common structures, preprocessor macros and defined types. Each function in Chapter 6, “Headers” specifies the headers that an application must include in order to use that function. In most cases only one header is required. These headers are present on an application development system; they do not have to be present on the target execution system.

<curses.h>

Name
curses.h - definitions for screen handling and optimization functions

Synopsis
#include <curses.h>

Description

Objects

The <curses.h> header provides a declaration for COLOR_PAIRS, COLORS, COLS, curscr, LINES and stdscr.

Constants

The following constants are defined:

EOF Function return value for end-of-file
ERR Function return value for failure
FALSE Boolean false value
OK Function return value for success
TRUE Boolean true value
WEOF Wide-character function return value for end-of-file, as defined in <wchar.h>.

The following constant is defined if the implementation supports the indicated revision of the X/Open Curses specification.

(1.e. this document).

Data Types

The following data types are defined through typedef:

attr_t An OR-ed set of attributes
bool  Boolean data type

chtype
A character, attributes and a color-pair

SCREEN
An opaque terminal representation

wchar_t
As described in <stddef.h>

wint_t  As described in <wchar.h>

cchar_t
References a string of wide characters

WINDOW
An opaque window representation

The inclusion of <curses.h> may make visible all symbols from the headers
<stdio.h>, <term.h>, <termios.h> and <wchar.h>.

Attribute Bits

The following symbolic constants are used to manipulate objects of type attr_t:

WA_ ALTCHEARSET
Alternate character set

WA_ BLINK
Blinking

WA_ BOLD
Extra bright or bold

WA_ DIM
Half bright

WA_ HORIZONTAL
Horizontal highlight

WA_ INVIS
Invisible

WA_ LEFT
Left highlight

WA_ LOW
Low highlight

WA_ PROTECT
Protected

WA_ REVERSE
Reverse video

WA_ RIGHT
Right highlight

WA_ STANDOUT
Best highlighting mode of the terminal

WA_ TOP
Top highlight
WA_UNDERLINE
   Underlining

WA_VERTICAL
   Vertical highlight

These attribute flags shall be distinct.

The following symbolic constants are used to manipulate attribute bits in objects of
type ctype:

A_ALTCHARSET
   Alternate character set

A_BLINK
   Blinking

A_BOLD
   Extra bright or bold

A_DIM
   Half bright

A_INVIS
   Invisible

A_PROTECT
   Protected

A_REVERSE
   Reverse video

A_STANDOUT
   Best highlighting mode of the terminal

A_UNDERLINE
   Underlining

These attribute flags need not be distinct except when _XOPEN_CURSES is
defined and the application sets _XOPEN_SOURCE_EXTENDED to 1.

The following symbolic constants can be used as bit-masks to extract the
components of a ctype:

A_ATTRIBUTES
   Bit-mask to extract attributes

A_CHARTEXT
   Bit-mask to extract a character

A_COLOR
   Bit-mask to extract color-pair information

The following symbolic constants can be used as bit-masks to extract the
components of a ctype:

A_ATTRIBUTES
   Bit-mask to extract attributes

A_CHARTEXT
   Bit-mask to extract a character

A_COLOR
   Bit-mask to extract color-pair information

Line-Drawing Constants
The `<curses.h>` header defines the symbolic constants shown in the leftmost two columns of the following table for use in drawing lines. The symbolic constants that begin with ACS_ are char constants. The symbolic constants that begin with WACS_ are cchar_t constants for use with the wide-character interfaces that take a pointer to a cchar_t.

In the POSIX locale, the characters shown in the POSIX Locale Default column are used when the terminal database does not specify a value using the acsc capability.

<table>
<thead>
<tr>
<th>char Constant</th>
<th>char_t Constant</th>
<th>POSIX Locale Default</th>
<th>Glyph Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACS_ULCORNER</td>
<td>WACS_ULCORNER</td>
<td>+</td>
<td>upper left-hand corner</td>
</tr>
<tr>
<td>ACS_LLCORNER</td>
<td>WACS_LLCORNER</td>
<td>+</td>
<td>lower left-hand corner</td>
</tr>
<tr>
<td>ACS_URCORNER</td>
<td>WACS_URCORNER</td>
<td>+</td>
<td>upper right-hand corner</td>
</tr>
<tr>
<td>ACS_LRCORNER</td>
<td>WACS_LRCORNER</td>
<td>+</td>
<td>lower right-hand corner</td>
</tr>
<tr>
<td>ACS_RTEE</td>
<td>WACS_RTEE</td>
<td>+</td>
<td>right tee (-</td>
</tr>
<tr>
<td>ACS_LTEE</td>
<td>WACS_LTEE</td>
<td>+</td>
<td>left tee (</td>
</tr>
<tr>
<td>ACS_BTEE</td>
<td>WACS_BTEE</td>
<td>+</td>
<td>bottom tee (</td>
</tr>
<tr>
<td>ACS_TTEE</td>
<td>WACS_TTEE</td>
<td>+</td>
<td>top tee (</td>
</tr>
<tr>
<td>ACS_HLINE</td>
<td>WACS_HLINE</td>
<td>-</td>
<td>horizontal line</td>
</tr>
<tr>
<td>ACS_VLINE</td>
<td>WACS_VLINE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACS_PLUS</td>
<td>WACS_PLUS+</td>
<td>plus</td>
<td></td>
</tr>
<tr>
<td>ACS_S1</td>
<td>WACS_S1</td>
<td>-</td>
<td>scan line 1</td>
</tr>
<tr>
<td>ACS_S9</td>
<td>WACS_S9</td>
<td>_</td>
<td>scan line 9</td>
</tr>
<tr>
<td>ACS_DIAMOND</td>
<td>WACS_DIAMOND</td>
<td>+</td>
<td>diamond</td>
</tr>
<tr>
<td>ACS_CKBOARD</td>
<td>WACS_CKBOARD</td>
<td>:</td>
<td>checker board (stipple)</td>
</tr>
<tr>
<td>ACS_DEGREE</td>
<td>WACS_DEGREE</td>
<td>'</td>
<td>degree symbol</td>
</tr>
<tr>
<td>ACS_PLMINUS</td>
<td>WACS_PLMINUS</td>
<td>#</td>
<td>plus/minus</td>
</tr>
<tr>
<td>ACS_BULLET</td>
<td>WACS_BULLET</td>
<td>o</td>
<td>bullet</td>
</tr>
<tr>
<td>ACS_LARROW</td>
<td>WACS_LARROW</td>
<td>&lt;</td>
<td>arrow pointing left</td>
</tr>
<tr>
<td>ACS_RARROW</td>
<td>WACS_RARROW</td>
<td>&gt;</td>
<td>arrow pointing right</td>
</tr>
<tr>
<td>ACS_DARROW</td>
<td>WACS_DARROW</td>
<td>v</td>
<td>arrow pointing down</td>
</tr>
<tr>
<td>ACS_UARROW</td>
<td>WACS_UARROW</td>
<td>^</td>
<td>arrow pointing up</td>
</tr>
<tr>
<td>ACS_BOARD</td>
<td>WACS_BOARD</td>
<td>#</td>
<td>board of squares</td>
</tr>
<tr>
<td>ACS_LANTERN</td>
<td>WACS_LANTERN</td>
<td>#</td>
<td>lantern symbol</td>
</tr>
<tr>
<td>ACS_BLOCK</td>
<td>WACS_BLOCK</td>
<td>#</td>
<td>solid square block</td>
</tr>
</tbody>
</table>

Color-Related Macros

The following color-related macros are defined:

COLOR_BLACK
COLOR_BLUE
COLOR_GREEN
COLOR_CYAN
COLOR_RED
COLOR_MAGENTA
COLOR_YELLOW
COLOR_WHITE

Coordinate-Related Macros
The following coordinate-related macros are defined:

```c
void getbegyx(WINDOW *win, int y, int x);
void getmaxyx(WINDOW *win, int y, int x);
void getparyx(WINDOW *win, int y, int x);
void getyx(WINDOW *win, int y, int x);
```

**Key Codes**

The following symbolic constants representing function key values are defined:

<table>
<thead>
<tr>
<th>Key Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_CODE_YES</td>
<td>Used to indicate that a wchar_t variable contains a key code</td>
</tr>
<tr>
<td>KEY_BREAK</td>
<td>Break key</td>
</tr>
<tr>
<td>KEY_DOWN</td>
<td>Down arrow key</td>
</tr>
<tr>
<td>KEY_UP</td>
<td>Up arrow key</td>
</tr>
<tr>
<td>KEY_LEFT</td>
<td>Left arrow key</td>
</tr>
<tr>
<td>KEY_RIGHT</td>
<td>Right arrow key</td>
</tr>
<tr>
<td>KEY_HOME</td>
<td>Home key</td>
</tr>
<tr>
<td>KEY_BACKSPACE</td>
<td>Backspace</td>
</tr>
<tr>
<td>KEY_F0</td>
<td>Function keys; space for 64 keys is reserved</td>
</tr>
<tr>
<td>KEY_F(n)</td>
<td>For 0_&lt;n_&lt;63</td>
</tr>
<tr>
<td>KEY_DL</td>
<td>Delete line</td>
</tr>
<tr>
<td>KEY_IL</td>
<td>Insert line</td>
</tr>
<tr>
<td>KEY_DC</td>
<td>Delete character</td>
</tr>
<tr>
<td>KEY_IC</td>
<td>Insert char or enter insert mode</td>
</tr>
<tr>
<td>KEY_EIC</td>
<td>Exit insert char mode</td>
</tr>
<tr>
<td>KEY_CLEAR</td>
<td>Clear screen</td>
</tr>
<tr>
<td>KEY_EOS</td>
<td>Clear to end of screen</td>
</tr>
<tr>
<td>KEY_EOL</td>
<td>Clear to end of line</td>
</tr>
<tr>
<td>KEY_SF</td>
<td>Scroll 1 line forward</td>
</tr>
<tr>
<td>KEY_SR</td>
<td>Scroll 1 line backward (reverse)</td>
</tr>
<tr>
<td>KEY_NPAGE</td>
<td>Next page</td>
</tr>
</tbody>
</table>
The virtual keypad is a 3-by-3 keypad arranged as follows:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>UP</td>
<td>A3</td>
</tr>
<tr>
<td>LEFT</td>
<td>B2</td>
<td>RIGHT</td>
</tr>
<tr>
<td>C1</td>
<td>DOWN</td>
<td>C3</td>
</tr>
</tbody>
</table>

Each legend, such as A1, corresponds to a symbolic constant for a key code from the preceding table, such as KEY_A1.

The following symbolic constants representing function key values are also defined:

<table>
<thead>
<tr>
<th>Key Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_BTAB</td>
<td>Back tab key</td>
</tr>
<tr>
<td>KEY_BEG</td>
<td>Beginning key</td>
</tr>
<tr>
<td>KEY_CANCEL</td>
<td>Cancel key</td>
</tr>
<tr>
<td>KEY_CLOSE</td>
<td>Close key</td>
</tr>
</tbody>
</table>
KEY_COMMAND
Cmd (command) key

KEY_COPY
Copy key

KEY_CREATE
Create key

KEY_END
End key

KEY_EXIT
Exit key

KEY_FIND
Find key

KEY_HELP
Help key

KEY_MARK
Mark key

KEY_MESSAGE
Message key

KEY_MOVE
Move key

KEY_NEXT
Next object key

KEY_OPEN
Open key

KEY_OPTIONS
Options key

KEY_PREVIOUS
Previous object key

KEY_REDO
Redo key

KEY_REFERENCE
Reference key

KEY_REFRESH
Refresh key

KEY_REPLACE
Replace key

KEY_RESTART
Restart key

KEY_RESUME
Resume key

KEY_SAVE
Save key

KEY_SBEG
Shifted beginning key
Curses

- **KEY_SCANCEL**
  - Shifted cancel key

- **KEY_SCOMMAND**
  - Shifted command key

- **KEY_SCOPY**
  - Shifted copy key

- **KEY_SCREATE**
  - Shifted create key

- **KEY_SDC**
  - Shifted delete char key

- **KEY_SDL**
  - Shifted delete line key

- **KEY_SELECT**
  - Select key

- **KEY_SEND**
  - Shifted end key

- **KEY_SEOL**
  - Shifted clear line key

- **KEY_SEXIT**
  - Shifted exit key

- **KEY_SFINDD**
  - Shifted find key

- **KEY_SHELP**
  - Shifted help key

- **KEY_SHOME**
  - Shifted home key

- **KEY_SIC**
  - Shifted input key

- **KEY_SLEFT**
  - Shifted left arrow key

- **KEY_SMESSAGE**
  - Shifted message key

- **KEY_SMOVE**
  - Shifted move key

- **KEY_SNEXT**
  - Shifted next key

- **KEY_SOPTIONS**
  - Shifted options key

- **KEY_SPREVIOUS**
  - Shifted prev key

- **KEY_SPRINT**
  - Shifted print key

- **KEY_SREDO**
  - Shifted redo key
Function Prototypes

The following are declared as functions, and may also be defined as macros:

```c
int addch(const chtype);
int addchstr(const chtype *, init);
int addchnstr(chtype *const chstr, int n);
int addchstr(const chtype *);
int addnstr(const char *, init);
int addstr(const char *);
int add_wch(const cchar_t *);
int add_wchnstr(const cchar_t *, int);
int add_wchstr(const cchar_t *);
int addwstr(const wchar_t *);
int attroff(int);
int attron(int);
int attrset(int);
int attr_get(attr_t *, short *, void*);
int attr_off(attr_t void *);
int attr_on(attr_t, void *);
int attr_set(attr_t, short, void *);
int baudrate(void);
int beep(void);
int bkgd(chtype);
void bkgdset(chtype);
ind bgkrd(const cchar_t *);
void bkgrndset(const cchar_t *);
int border(chtype, chtype, chtype, chtype, chtype,
          chtype, chtype, chtype);
int border_set(const cchar_t *,
               const cchar_t *,
               const cchar_t *,
               const cchar_t *,
               const cchar_t *,
               const cchar_t *,
               const cchar_t *,
               const cchar_t *);
int box(WINDOW *, chtype, chtype);
int box_set(WINDOW *, const cchar_t *, const cchar_t *);
bool can_change_color(void);
int cbreak(void);
int chgat(int, attr_t *, short, const void *);
int clearok(WINDOW *, bool);
int clear(void);
int clrtoeol(void);
```
CURSES

int color_content(short, short *, short *, short *);
int COLOR_PAIR(int);
int Color_set(short, void *);
int copywin(const WINDOW *, WINDOW *, int, int, int,
             int, int, int, int);
int curs_set(int);
int def_prog_mode(void);
int def_shell_mode(void);
int delay_output(int);
int delch(void);
int deleteln(void);
void delscreen(SCREEN *);  
int delwin(WINDOW *);
WINDOW *derwin(WINDOW *, int, int, int, int,
               int, int, int, int);
int doupdate(void);
WINDOW *dupwin(WINDOW *);
int echo(void);
int echochar(const chtype);
int echo_wchar(const cchar_t *);
int endwin(void);
int erasechar(void);
int erase(void);
int erasewchar(wchar_t *);
void filter(void);
int flash(void);
int flushinp(void);
chtype getbkgd(WINDOW *);
int getbgrnd(cchar_t *);
int getcchar(const cchar_t *, wchar_t *, attr_t *,
              short *, void *);
int getch(void);
int getnstr(char *, int);
int getwnstr(wint_t *, int);
int getstr(char *);
int get_wch(wint_t *);
WINDOW *getwin(FILE *);
int get_wstr(wint_t *);
int halfdelay(int);
bool has_colors(void);
bool has_ic(void);
bool has_il(void);
int hline(chtype, int);
int hline_set(const cchar_t *, int);
void idcok(WINDOW *, bool);
int idlok(WINDOW *, win, bool bf);
void immedok(WINDOW *, bool);
chtype inch(void);
int inchnstr(chtype *, int);
int inchstr(chtype *);  
WINDOW *initscr(void);
int init_color(short, short, short, short);
int init_pair(short, short, short);
int innstr(char *, int);
int innwstr(wchar_t *, int);
int insch(chtype);
int insdelln(int);
int insertln(void);
intinsnstr(const char *, int);
int insnstr(const wchar_t *, int);
int instr(char *);
int ins_wch(const char_t *);
int ins_wchstr(const char_t *);
int intrflush(WINDOW *, bool);
int in_wch(char_t *);
int in_wchnstr(cchar_t *, int);
int in_wchstr(const char *);
int inwstr(const char *);
bool isendwin(void);
bool is_linetouched(WINDOW *, int);
bool is_wintouched(WINDOW *);
char *keyname(int);
char *key_name(wchar_t);
int keypad(WINDOW *, bool);
char killchar(void);
int killwchar(wchar_t *);
int leaveok(WINDOW *, bool);
char *longname(void);
int meta(WINDOW *, bool);
int move(int, int);
int mvaddch(int, int, const chtype);
int mvaddchnstr(int, int, const chtype *, int);
int mvaddchstr(int, int, const chtype *);
int mvaddnstr(int, int, const char *, int);
int mvaddwstr(int, int, wchar_t *, int);
int mvaddstr(int, int, const cchar_t *);
int mvadd_wch(int, int, const cchar_t *);
int mvadd_wchnstr(int, int, const cchar_t *, int);
int mvadd_wchstr(int, int, const cchar_t *);
int mvaddwstr(int, int, const wchar_t *);
int mvchgat(int, int, short, const void *);
int mvcur(int, int, int);
int mvdelsch(int, int);
int mvderwin(WINDOW *, int, int);
int mvgetnstr(int, int, char *, int);
int mvgetn_wstr(int, int, wint_t *, int);
int mvgetstr(int, int, char *);
int mvget_wch(int, int, wint_t *);
int mvget_wstr(int, int, wint_t *);
int mwhline(int, int, chtype, int);
int mwhline_set(int, int, const cchar_t *, int);
chtie mwinch(int, int);
int mwinchnstr(int, int, chtype *, int);
int mwinchnstr(int, int, chtype *);
int mvinnstr(int, int, wchar_t *, int);
int mvinsch(int, int, chtype);
int mvinsnstr(int, int, const char *, int);
in mvins_nwstr(int, int, const wchar_t *, int);
in mvinsstr(int, int, const char *);
in mvinsnstr(int, int, const char *);
in mvins_wch(int, int, const cchar_t *);
in mvins_watn(int, int, const wchar_t *

in mvwaddch(WINDOW *, int, int, const chtype);
in mvwaddchnstr(WINDOW *, int, int, const chtype *, init);
in mvwaddchstr(WINDOW *, int, int, const chtype *);
in mvwaddnstr(WINDOW *, int, int, const char *, int);
in mvwaddwstr(WINDOW *, int, int, const wchar_t *, int);
in mvwaddstr(WINDOW *, int, int, const cchar_t *);
in mvwadd_wch(WINDOW *, int, int, const cchar_t *);
in mvwadd_wchnstr(WINDOW *, int, int, const cchar_t *, int);
in mvwadd_wchstr(WINDOW *, int, int, const cchar_t *);
in mvwaddwstr(WINDOW *, int, int, const wchar_t *);
in mvwchgat(WINDOW *, int, int, int, attr_t, ...,
CURSES

short, const void *);
int mvwdelch(WINDOW *, int, int);
int mvwgetch(WINDOW *, int, int);
int mvwgetnstr(WINDOW *, int, int, char *, int);
int mvwgetn_wstr(WINDOW *, int, int, wchar_t *, int);
int mvwgetstr(WINDOW *, int, int, char *);
int mvwget_wch(WINDOW *, int, int, wchar_t *);
int mvwget_wstr(WINDOW *, int, int, wchar_t *);
int mvwhline(WINDOW *, int, int, chtype, int);
int mvwhline_set(WINDOW *, int, int, chtype, int);
int mvwin(WINDOW *, int, int);
chtype mvwinch(WINDOW *, int, int);
int mvwinchnstr(WINDOW *, int, int, chtype *, int);
int mvwinnstr(WINDOW *, int, int, char *, int);
int mvwinnwstr(WINDOW *, int, int, wchar_t *, int);
int mvwinsch(WINDOW *, int, int, chtype);
int mvwinsnstr(WINDOW *, int, int, const char *, int);
int mvwinsnstr(WINDOW *, int, int, wchar_t *, int);
int mvwins_nwstr(WINDOW *, int, int, const wchar_t *, int);
int mvwinsnstr(WINDOW *, int, int, const char *, int);
int mvwinsch(WINDOW *, int, int, chtype);
int mvwinnstr(WINDOW *, int, int, char *, int);
int mvwinnwstr(WINDOW *, int, int, wchar_t *, int);
int mvwins_chstr(WINDOW *, int, int, const cchar_t *);
int mvwinnstr(WINDOW *, int, int, const wchar_t *);
int mvwinsnstr(WINDOW *, int, int, wchar_t *);
int mvwinsnstr(WINDOW *, int, int, char *, ...);
int mvwsscanf(WINDOW *, int, int, char *, ...);
int mvwvline(WINDOW *, int, int, chtype, int);
int mvwvline_set(WINDOW *, int, int, chtype, int);
int napms(int);
WINDOW *newpad(int, int);
SCREEN *newterm(char *, FILE *, FILE *);
WINDOW *newwin(int, int, int, int);
int nl(void);
int nocbreak(void);
int nodelay(WINDOW *, bool);
int noecho(void);
int nonl(void);
void nqiflush(void);
int noraw(void);
int notimeout(WINDOW *, bool);
int overlay(const WINDOW *, WINDOW *);
int overwrite(const WINDOW *, WINDOW *);
int pair_content(short, short *, short *);
int PAIR_NUMBER(int);
int pechochar(WINDOW *, chtype);
int pecho_wchar(WINDOW *, const cchar_t *);
int pnoutrefresh(WINDOW *, int, int, int, int, int);
int prefresh(WINDOW *, int, int, int, int, int);
int printw(char *, ...);
int putp(const cchar_t *);
int putwin(WINDOW *, FILE *);
void qiflush(void);
int raw(void);
int redrawwin(WINDOW *);
int refresh(void);
int resetty(void);
int reset_prog_mode(void);
int reset_shell_mode(void);
int resetty(void);
int ripoffline(int, int (*)(WINDOW *, int));
int savetty(void);
int scanw(char *, ...);
int scr_dump(const char *);
int scr_init(const char *);
int waddwstr(WINDOW *, const wchar_t *);
int wattroff(WINDOW *, int);
int wattron(WINDOW *, int);
int wattrset(WINDOW *, int);
int wattr_get(WINDOW *, attr_t *, short *, void *);
int wattr_off(WINDOW *, attr_t void);
int wattr_on(WINDOW *, attr_t void);
int wattr_set(WINDOW *, attr_t, short, void *
int wbkgd(WINDOW *, chtype);
void wbkgdset(WINDOW *, chtype);
int wbkgnd(WINDOW *, const cchar_t *);
void wbkgndset(WINDOW *, const cchar_t *);
int wborder(WINDOW *, chtype, chtype, chtype, chtype,
           chtype, chtype, chtype, chtype);
int wborder_set(WINDOW *, const cchar_t *, const cchar_t *,
                const cchar_t *,
                const cchar_t *,
                const cchar_t *,
                const cchar_t *,
                const cchar_t *);
int wchgat(WINDOW *, int, attr_t, short, const void *);
int wclear(WINDOW *);
int wcirtofbot(WINDOW *);
int wcirtoeof(WINDOW *);
void wcursyncup(WINDOW *);
int wcolor_set(WINDOW *, short, void *);
in dutelch(WINDOW *);
int wdelete1n(WINDOW *);
int wechachar(WINDOW *, const chtype);
int wecho_wchar(WINDOW *, const chtype *);
int wclear(WINDOW *);
int wclrtobot(WINDOW *);
int wclrtoeol(WINDOW *);
void wcursyncup(WINDOW *);
int wcolor_set(WINDOW *, short, void *);
in dutelch(WINDOW *);
int wdelete1n(WINDOW *);
int wechachar(WINDOW *, const chtype);
int wecho_wchar(WINDOW *, const chtype *);
int wclear(WINDOW *);
int wclrtobot(WINDOW *);
int wclrtoeol(WINDOW *);
void wcursyncup(WINDOW *);
int wcolor_set(WINDOW *, short, void *);
in dutelch(WINDOW *);
int wdelete1n(WINDOW *);
int wechachar(WINDOW *, const chtype);
int wecho_wchar(WINDOW *, const chtype *);
int wclear(WINDOW *);
int wclrtobot(WINDOW *);
int wclrtoeol(WINDOW *);
void wcursyncup(WINDOW *);
int wcolor_set(WINDOW *, short, void *);
in dutelch(WINDOW *);
int wdelete1n(WINDOW *);
int wechachar(WINDOW *, const chtype);
int wecho_wchar(WINDOW *, const chtype *);
int wclear(WINDOW *);
int wclrtobot(WINDOW *);
int wclrtoeol(WINDOW *);
void wcursyncup(WINDOW *);
int wcolor_set(WINDOW *, short, void *);
in dutelch(WINDOW *);
int wdelete1n(WINDOW *);
int wechachar(WINDOW *, const chtype);
int wecho_wchar(WINDOW *, const chtype *);
int wclear(WINDOW *);
int wclrtobot(WINDOW *);
int wclrtoeol(WINDOW *);
void wcursyncup(WINDOW *);
void wsyncdown(WINDOW *);
void wtimeout(WINDOW *, int);
int wtouchln(WINDOW *, int, int, int);
wchar_t *wunctrl(cchar_t *);
int wvline(WINDOW *, chtype, int);
int wvline_set(WINDOW *, const cchar_t *, int);

See Also

<stdio.h>, <term.h>, <termios.h>, <unctrl.h>, <wchar.h>.

_NAME_ term.h - terminal capabilities

SYNOPSIS

#include <term.h>

DESCRIPTION

The following data type is defined through _typedef_:

TERMINAL

An opaque representation of the capabilities for a single terminal from the
terminfo database.

The <term.h> header provides a declaration for the following object: _cur_term_. It
represents the current terminal record from the terminfo database that the
application has selected by calling _set_curterm_().

The <term.h> header contains the variable names listed in the Variable column.

The following are declared as functions, and may also be defined as macros:

int del_curterm(TERMINAL *);
int putp(const char *);
int restartterm(char *, int, int *);
TERMINAL *set_curterm(TERMINAL *);
int setupterm(char *, int, int *);
int tgetent(char *, const char);
int tgetflag(char *);
int tgetnum(char *);
char *tgetstr(char *, char **);
char *tgoto(char *, int, int);
int tigetflag(char *);
int tigetnum(char *);
char *tigetstr(char *);
char *tparm(char *, long, long, long, long, long,
            long, long, long, long);
int tputs(const char *, int, int (*)(int));

See Also

printf(), putp(), tigetflag(), tgetent(), <curses.h>.
<unctrlh>

Name
unctrl.h - definitions for unctrl()

Description
The <unctrl.h> header defines the chtype type as defined in <curses.h>.

The following is declared as a function, and may also be defined as a macro:
char *unctrl(chtype);

See Also
unctrl(), <curses.h>.
Chapter 7. Terinfo Source Format (ENHANCED CURSES)

The terinfo database contains a description of the capabilities of a variety of devices, such as terminals and printers. Devices are described by specifying a set of capabilities, by quantifying certain aspects of the device, and by specifying character sequences that effect particular results.

This chapter specifies the format of terinfo source files.

X/Open-compliant implementations provide a facility that accepts source files in the format specified in this chapter as a means of entering information into the terinfo database. The facility for installing this information into the database is implementation-specific. A valid terinfo entry describing a given model of terminal can be added to terinfo on any X/Open-compliant implementation to permit use of the same terminal model.

The terinfo database is often used by screen-oriented applications such as vi and Curses programs, as well as by some utilities such as ls and more. This usage allows them to work with a variety of devices without changes to the programs.

Source File Syntax

Source files can use the ISO 8859-1 codeset. The behavior when the source file is in another codeset is unspecified. Traditional practice has been to translate information from other codesets into the source file syntax.

Terinfo source files consist of one or more device descriptions. Each description defines a mnemonic name for the terminal model. 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 terinfo source file must end in a comma. Every line in a terinfo source file except the header must be indented with one or more white spaces (either spaces or tabs).

Entries in terinfo 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 terinfo source file:

```
alias1 | alias2 | ... | aliasn | 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 terinfo database and they must conform to file naming conventions established by implementation-specific terinfo compilation utilities. Implementations will recognize alias names consisting only of characters from the portable filename character set except that implementations need not accept a first character of minus(-). For example, a typical restriction is

© Copyright IBM Corp. 1996, 2013
that they cannot contain white space or slashes. There may be further constraints imposed on source file values by the implementation-specific terminfo compilation utilities.

Each capability in terminfo is of one of the following types:

- Boolean capabilities show that a device has or does not have a particular feature.
- Numeric capabilities quantify particular features of a device.
- String capabilities provide sequences that can be used to perform particular operations on devices.

Capability names adhere to an informal length limit of five characters. 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 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_. These capabilities are defined in <term.h>.

### Minimum Guaranteed Limits

All X/Open-compliant implementations support at least the following limits for the terminfo source file:

<table>
<thead>
<tr>
<th>Source File Characteristic</th>
<th>Minimum Guaranteed Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of a line</td>
<td>1023 bytes</td>
</tr>
<tr>
<td>Length of a terminal alias</td>
<td>14 bytes</td>
</tr>
<tr>
<td>Length of a terminal model name</td>
<td>128 bytes</td>
</tr>
<tr>
<td>Width of a single field</td>
<td>128 bytes</td>
</tr>
<tr>
<td>Length of a string value</td>
<td>1000 bytes</td>
</tr>
<tr>
<td>Length of a string representing a numeric value</td>
<td>99 digits</td>
</tr>
<tr>
<td>Magnitude of a numeric value</td>
<td>0 up to and including 32767</td>
</tr>
</tbody>
</table>

An implementation may support higher limits than those specified above.

### Formal Grammar

The grammar and lexical conventions in this section together describe the syntax for terminfo terminal descriptions within a terminfo source file. A terminal description that satisfies the requirements of this section will be accepted by all implementations.

```
descriptions : START_OF_HEADER_LINE rest_of_header_line feature_lines
               | descriptions START_OF_HEADER_LINE rest_of_header_line
               | feature_lines
               ;

rest_of_header_line : PIPE LONGNAME COMMA NEWLINE
                     | aliases PIPE LONGNAME COMMA NEWLINE
                     ;
```
The lexical conventions for *terminfo* descriptions are as follows:

1. White space consists of the ‘ ’ and <tab> character.
2. An ALIAS may contain any graph \(^8\) characters other than ‘,’ ‘/’ and ‘|’.
3. A LONGNAME may contain any print \(^9\) characters other than ‘,’ and ‘|’.
4. A BOOLEAN feature may contain any print characters other than ‘,’ ‘=’, and ‘#’.
5. A NUMERIC feature consists of:
   a. A name which may contain any print character other than ‘,’ ‘=’, and ‘#’.
   b. The ‘#’ character.
   c. A positive integer which conforms to the C language convention for integer constants.
6. A STRING feature consists of:
   a. A name which may contain any print character other than ‘,’ ‘=’, and ‘#’.
   b. The ‘=’ character.
   c. A string which may contain any print characters other than ‘,’.
7. White space immediately following a ‘,’ is ignored.
8. Comments consist of <bol>, optional whitespace, a required ‘#’, and a terminating <eol>.
9. A header line must begin in column one.

---

4. An ALIAS that begins in column one. This is handled by the lexical analyzer.
5. A BOOLEAN feature that begins after column one but is the first feature on the feature line. This is handled by the lexical analyzer.
6. A NUMERIC feature that begins after column one but is the first feature on the feature line. This is handled by the lexical analyzer.
7. A STRING feature that begins after column one but is the first feature on the feature line. This is handled by the lexical analyzer.
8. Graph characters are those characters for which `isgraph()` returns non-zero.
9. Print characters are those characters for which `isprint()` returns non-zero.
10. A feature line must not begin in column one.
11. Blank lines are ignored.

**Defined Capabilities**

X/Open defines the capabilities listed in the following table. All X/Open-compliant implementations must accept each of these capabilities in an entry in a `terminfo` source file. Implementations use this information to determine how properly to operate the current terminal. In addition, implementations return any of the current terminal’s capabilities when the application calls the query functions listed in `tgetent()`.

The table of capabilities has the following columns:

**Variable**
Names for use by the Curses functions that operate on the `terminfo` database. These names are reserved and the application must not define them.

**Capname**
The short name for a capability specified in the `terminfo` source file. It is used for updating the source file and by the `tput` command.

**Termcap**
Codes provided for compatibility with older applications. These codes are TO BE WITHDRAWN. Because of this, not all Capnames have Termcap codes.

### Booleans

<table>
<thead>
<tr>
<th>Variable</th>
<th>Capname</th>
<th>Termcap</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_left_margin</td>
<td>bw</td>
<td>bw</td>
<td><code>cub1</code> wraps from column 0 to last column</td>
</tr>
<tr>
<td>auto_right_margin</td>
<td>am</td>
<td>am</td>
<td>Terminal has automatic margins</td>
</tr>
<tr>
<td>back_color_erase</td>
<td>bce</td>
<td>ut</td>
<td>Screen erased with background color</td>
</tr>
<tr>
<td>can_change</td>
<td>ccc</td>
<td>cc</td>
<td>Terminal can re-define existing color</td>
</tr>
<tr>
<td>ceol_standout_glitch</td>
<td>xhp</td>
<td>xs</td>
<td>Standout not erased by overwriting (hp)</td>
</tr>
<tr>
<td>col_addr_glitch</td>
<td>xhpa</td>
<td>YA</td>
<td>Only positive motion for <code>hpa/mhpa</code> caps</td>
</tr>
<tr>
<td>cpi_changes_res</td>
<td>cpix</td>
<td>YF</td>
<td>Changing character pitch changes resolution</td>
</tr>
<tr>
<td>cr_cancels_micro_mode</td>
<td>crxm</td>
<td>YB</td>
<td>Using <code>cr</code> turns off micro mode</td>
</tr>
<tr>
<td>dest_tabs_magic_smso</td>
<td>xt</td>
<td>xt</td>
<td>Destructive tabs, magic <code>smso</code> char (11061)</td>
</tr>
<tr>
<td>eat_newline_glitch</td>
<td>xenl</td>
<td>xn</td>
<td>Newline ignored after 80 columns (Concept)</td>
</tr>
<tr>
<td>erase_overstrike</td>
<td>eo</td>
<td>eo</td>
<td>Can erase overstrikes with a blank</td>
</tr>
<tr>
<td>generic_type</td>
<td>gn</td>
<td>gn</td>
<td>Generic line type (e.g., dialup, switch)</td>
</tr>
<tr>
<td>hard_copy</td>
<td>hc</td>
<td>hc</td>
<td>Hardcopy terminal</td>
</tr>
<tr>
<td>hard_cursor</td>
<td>chts</td>
<td>HC</td>
<td>Cursor is hard to see</td>
</tr>
<tr>
<td>has_meta_key</td>
<td>km</td>
<td>km</td>
<td>Has a meta key (shift, sets parity bit)</td>
</tr>
<tr>
<td>has_print_wheel</td>
<td>daisy</td>
<td>YC</td>
<td>Printer needs operator to change character set</td>
</tr>
<tr>
<td>has_status_line</td>
<td>hs</td>
<td>hs</td>
<td>Has extra “status line”</td>
</tr>
<tr>
<td>Variable</td>
<td>Capname</td>
<td>Termcap</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>---------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>hue_lightness_saturation</td>
<td>hls</td>
<td>h1</td>
<td>Terminal uses only HLS color notation (Tektronix)</td>
</tr>
<tr>
<td>insert_null_glitch</td>
<td>in</td>
<td>in</td>
<td>Insert mode distinguishes nulls</td>
</tr>
<tr>
<td>lpi_changes_res</td>
<td>lpix</td>
<td>YG</td>
<td>Changing line pitch changes resolution</td>
</tr>
<tr>
<td>memory_above</td>
<td>da</td>
<td>da</td>
<td>Display may be retained above the screen</td>
</tr>
<tr>
<td>memory_below</td>
<td>db</td>
<td>db</td>
<td>Display may be retained below the screen</td>
</tr>
<tr>
<td>move_insert_mode</td>
<td>mir</td>
<td>mi</td>
<td>Safe to move while in insert mode</td>
</tr>
<tr>
<td>move_standout_mode</td>
<td>msgr</td>
<td>ms</td>
<td>Safe to move in standout modes</td>
</tr>
<tr>
<td>needs_xon_xoff</td>
<td>nxon</td>
<td>nx</td>
<td>Padding won't work, xon/xoff required</td>
</tr>
<tr>
<td>no_esc_ctlc</td>
<td>xsb</td>
<td>xb</td>
<td>Beehive (f1=escape, f2=ctrl C)</td>
</tr>
<tr>
<td>no_pad_char</td>
<td>npc</td>
<td>NP</td>
<td>Pad character doesn't exist</td>
</tr>
<tr>
<td>non_dest_scroll_region</td>
<td>ndscr</td>
<td>ND</td>
<td>Scrolling region is nondestructive</td>
</tr>
<tr>
<td>non_rev_rmcup</td>
<td>nrrmc</td>
<td>NR</td>
<td>smcup does not reverse</td>
</tr>
<tr>
<td>over_strike</td>
<td>os</td>
<td>os</td>
<td>Terminal overstrikes on hard-copy terminal</td>
</tr>
<tr>
<td>prtr_silent</td>
<td>mc5i</td>
<td>5i</td>
<td>Printer won't echo on screen</td>
</tr>
<tr>
<td>row_addr_glitch</td>
<td>xvpa</td>
<td>YD</td>
<td>Only positive motion for vpa/mvpa caps</td>
</tr>
<tr>
<td>semi_auto_right_margin</td>
<td>sam</td>
<td>YE</td>
<td>Printing in last column causes cr</td>
</tr>
<tr>
<td>status_line_esc_ok</td>
<td>eslok</td>
<td>es</td>
<td>Escape can be used on the status line</td>
</tr>
<tr>
<td>tilda_gitch</td>
<td>hz</td>
<td>hz</td>
<td>Hazeltine; can't print tilde (˜)</td>
</tr>
<tr>
<td>transparent_underline</td>
<td>ul</td>
<td>ul</td>
<td>Underline character overstrikes</td>
</tr>
<tr>
<td>xon_xoff</td>
<td>xon</td>
<td>xo</td>
<td>Terminal uses xon/xoff handshaking</td>
</tr>
</tbody>
</table>

**Numbers**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Capname</th>
<th>Termcap</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit_image_entwining</td>
<td>bitwin</td>
<td>Yo</td>
<td>Number of passes for each bit-map row</td>
</tr>
<tr>
<td>bit_image_type</td>
<td>bitype</td>
<td>Yp</td>
<td>Type of bit image device</td>
</tr>
<tr>
<td>buffer_capacity</td>
<td>bfsz</td>
<td>Ya</td>
<td>Number of bytes buffered before printing</td>
</tr>
<tr>
<td>buttons</td>
<td>btns</td>
<td>BT</td>
<td>Number of buttons on the mouse</td>
</tr>
<tr>
<td>columns</td>
<td>cols</td>
<td>co</td>
<td>Number of columns in a line</td>
</tr>
<tr>
<td>dot_horz_spacing</td>
<td>spinh</td>
<td>Yc</td>
<td>Spacing of dots horizontally in dots per inch</td>
</tr>
<tr>
<td>dot_vert_spacing</td>
<td>spinv</td>
<td>Yb</td>
<td>Spacing of pins vertically in pins per inch</td>
</tr>
<tr>
<td>init_tabs</td>
<td>it</td>
<td>it</td>
<td>Tabs initially every # spaces</td>
</tr>
<tr>
<td>label_height</td>
<td>lh</td>
<td>lh</td>
<td>Number of rows in each label</td>
</tr>
<tr>
<td>label_width</td>
<td>lw</td>
<td>lw</td>
<td>Number of columns in each label</td>
</tr>
<tr>
<td>lines</td>
<td>lines</td>
<td>li</td>
<td>Number of lines on a screen or a page</td>
</tr>
<tr>
<td>lines_of_memory</td>
<td>lm</td>
<td>lm</td>
<td>Lines of memory if &gt; lines; 0 means varies</td>
</tr>
<tr>
<td>max_attributes</td>
<td>ma</td>
<td>ma</td>
<td>Maximum combined video attributes terminal can display</td>
</tr>
<tr>
<td>magic_cookie_glitch</td>
<td>xmc</td>
<td>sg</td>
<td>Number of blank characters left by smso or rmso</td>
</tr>
</tbody>
</table>

Chapter 7. Termino Source Format (ENHANCED CURSES) 145
### Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Capname</th>
<th>Termcap</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_colors</td>
<td>colors</td>
<td>Co</td>
<td>Maximum number of colors on the screen</td>
</tr>
<tr>
<td>max_micro_address</td>
<td>maddr</td>
<td>Yd</td>
<td>Maximum value in <code>micro_address</code></td>
</tr>
<tr>
<td>max_micro_jump</td>
<td>mjump</td>
<td>Ye</td>
<td>Maximum value in <code>parm_micro</code></td>
</tr>
<tr>
<td>max_pairs</td>
<td>pairs</td>
<td>pa</td>
<td>Maximum number of color-pairs on the screen</td>
</tr>
<tr>
<td>maximum_windows</td>
<td>wnum</td>
<td>MW</td>
<td>Maximum number of definable windows</td>
</tr>
<tr>
<td>micro_col_size</td>
<td>mcs</td>
<td>Yf</td>
<td>Character step size when in micro mode</td>
</tr>
<tr>
<td>micro_line_size</td>
<td>mls</td>
<td>Yg</td>
<td>Line step size when in micro mode</td>
</tr>
<tr>
<td>no_color_video</td>
<td>ncv</td>
<td>NC</td>
<td>Video attributes that can't be used with colors</td>
</tr>
<tr>
<td>num_labels</td>
<td>nlab</td>
<td>Nl</td>
<td>Number of labels on screen (start at 1)</td>
</tr>
<tr>
<td>number_of_pins</td>
<td>npins</td>
<td>Yh</td>
<td>Number of pins in print-head</td>
</tr>
<tr>
<td>output_res_char</td>
<td>orc</td>
<td>Yi</td>
<td>Horizontal resolution in units per character</td>
</tr>
<tr>
<td>output_res_line</td>
<td>orl</td>
<td>Yj</td>
<td>Vertical resolution in units per line</td>
</tr>
<tr>
<td>output_res_horz_inch</td>
<td>orhi</td>
<td>Yk</td>
<td>Horizontal resolution in units per inch</td>
</tr>
<tr>
<td>output_res_vert_inch</td>
<td>orvi</td>
<td>Yl</td>
<td>Vertical resolution in units per inch</td>
</tr>
<tr>
<td>padding_baud_rate</td>
<td>pb</td>
<td>pb</td>
<td>Lowest baud rate where padding needed</td>
</tr>
<tr>
<td>print_rate</td>
<td>cps</td>
<td>Ym</td>
<td>Print rate in characters per second</td>
</tr>
<tr>
<td>virtual_terminal</td>
<td>vt</td>
<td>vt</td>
<td>Virtual terminal number</td>
</tr>
<tr>
<td>wide_char_size</td>
<td>widcs</td>
<td>Yn</td>
<td>Character step size when in double-wide mode</td>
</tr>
<tr>
<td>width_status_line</td>
<td>wsl</td>
<td>ws</td>
<td>Number of columns in status line</td>
</tr>
</tbody>
</table>

### Strings

<table>
<thead>
<tr>
<th>Variable</th>
<th>Capname</th>
<th>Termcap</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acs_chars</td>
<td>acsc</td>
<td>ac</td>
<td>Graphic charset pairs aAbBcC</td>
</tr>
<tr>
<td>alt_scancode_esc</td>
<td>scesa</td>
<td>SB</td>
<td>Alternate escape for scancode emulation (default is for VT100)</td>
</tr>
<tr>
<td>back_tab</td>
<td>cbt</td>
<td>bt</td>
<td>Back tab</td>
</tr>
<tr>
<td>bell</td>
<td>bel</td>
<td>b1</td>
<td>Audible signal (bell)</td>
</tr>
<tr>
<td>bit_image_carriage_return</td>
<td>bircr</td>
<td>Yv</td>
<td>Move to beginning of same row</td>
</tr>
<tr>
<td>bit_image_newline</td>
<td>binel</td>
<td>Zz</td>
<td>Move to next row of the bit image</td>
</tr>
<tr>
<td>bit_image_repeat</td>
<td>birep</td>
<td>Xy</td>
<td>Repeat bit-image cell #1 #2 times</td>
</tr>
<tr>
<td>carriage_return</td>
<td>cr</td>
<td>cr</td>
<td>Carriage return</td>
</tr>
<tr>
<td>change_char_pitch</td>
<td>cpi</td>
<td>ZA</td>
<td>Change number of characters per inch</td>
</tr>
<tr>
<td>change_line_pitch</td>
<td>lpi</td>
<td>ZB</td>
<td>Change number of lines per inch</td>
</tr>
<tr>
<td>change_res_horz</td>
<td>chr</td>
<td>ZC</td>
<td>Change horizontal resolution</td>
</tr>
<tr>
<td>change_res_vert</td>
<td>cvr</td>
<td>ZD</td>
<td>Change vertical resolution</td>
</tr>
<tr>
<td>change_scroll_region</td>
<td>csr</td>
<td>cs</td>
<td>Change to lines #1 through #2 (VT100)</td>
</tr>
<tr>
<td>char_padding</td>
<td>rmp</td>
<td>rP</td>
<td>Like <code>ip</code> but when in replace mode</td>
</tr>
<tr>
<td>char_set_names</td>
<td>csnm</td>
<td>Zy</td>
<td>Returns a list of character set names</td>
</tr>
<tr>
<td>clear_all_tabs</td>
<td>tbc</td>
<td>ct</td>
<td>Clear all tab stops</td>
</tr>
<tr>
<td>clear_margins</td>
<td>mgc</td>
<td>MC</td>
<td>Clear all margins (top, bottom, and sides)</td>
</tr>
<tr>
<td>clear_screen</td>
<td>clear</td>
<td>cl</td>
<td>Clear screen and home cursor</td>
</tr>
<tr>
<td>Variable</td>
<td>Capname</td>
<td>Termcap</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>clr_bol</td>
<td>el1</td>
<td>cb</td>
<td>Clear to beginning of line, inclusive</td>
</tr>
<tr>
<td>clr_eol</td>
<td>el</td>
<td>ce</td>
<td>Clear to end of line</td>
</tr>
<tr>
<td>clr_eos</td>
<td>ed</td>
<td>cd</td>
<td>Clear to end of display</td>
</tr>
<tr>
<td>code_set_init</td>
<td>csin</td>
<td>c1</td>
<td>Init sequence for multiple codesets</td>
</tr>
<tr>
<td>color_names</td>
<td>colornm</td>
<td>Yw</td>
<td>Give name for color #1</td>
</tr>
<tr>
<td>column_address</td>
<td>hpa</td>
<td>ch</td>
<td>Set horizontal position to absolute #1</td>
</tr>
<tr>
<td>command_character</td>
<td>cmdch</td>
<td>CC</td>
<td>Terminal settable cmd character in prototype</td>
</tr>
<tr>
<td>create_window</td>
<td>cwin</td>
<td>CW</td>
<td>Define win #1 to go from #2,#3 to #4,#5</td>
</tr>
<tr>
<td>cursor_address</td>
<td>cup</td>
<td>cm</td>
<td>Move to row #1 col #2</td>
</tr>
<tr>
<td>cursor_down</td>
<td>cud1</td>
<td>do</td>
<td>Down one line</td>
</tr>
<tr>
<td>cursor_home</td>
<td>home</td>
<td>ho</td>
<td>Home cursor (if no cup)</td>
</tr>
<tr>
<td>cursor_invisible</td>
<td>civis</td>
<td>vi</td>
<td>Make cursor invisible</td>
</tr>
<tr>
<td>cursor_left</td>
<td>cub1</td>
<td>le</td>
<td>Move left one space.</td>
</tr>
<tr>
<td>cursor_mem_address</td>
<td>mr Cup</td>
<td>CM</td>
<td>Memory relative cursor addressing</td>
</tr>
<tr>
<td>cursor_normal</td>
<td>cnorm</td>
<td>ve</td>
<td>Make cursor appear normal (undo vs/vi)</td>
</tr>
<tr>
<td>cursor_right</td>
<td>cuff1</td>
<td>nd</td>
<td>Non-destructive space (cursor or carriage right)</td>
</tr>
<tr>
<td>cursor_to_ll</td>
<td>ll</td>
<td>l1</td>
<td>Last line, first column (if no cup)</td>
</tr>
<tr>
<td>cursor_up</td>
<td>cuu1</td>
<td>up</td>
<td>Upline (cursor up)</td>
</tr>
<tr>
<td>cursor_visible</td>
<td>cvvis</td>
<td>vs</td>
<td>Make cursor very visible</td>
</tr>
<tr>
<td>define_bit_image_region</td>
<td>defbi</td>
<td>Yx</td>
<td>Define rectangular bit-image region</td>
</tr>
<tr>
<td>define_char</td>
<td>defc</td>
<td>ZE</td>
<td>Define a character in a character set</td>
</tr>
<tr>
<td>delete_character</td>
<td>dch1</td>
<td>dc</td>
<td>Delete character</td>
</tr>
<tr>
<td>delete_line</td>
<td>d1</td>
<td>dl</td>
<td>Delete line</td>
</tr>
<tr>
<td>device_type</td>
<td>devt</td>
<td>dv</td>
<td>Indicate language/codeset support</td>
</tr>
<tr>
<td>dial_phone</td>
<td>dial</td>
<td>DI</td>
<td>Dial phone number #1</td>
</tr>
<tr>
<td>dis_status_line</td>
<td>dsl</td>
<td>ds</td>
<td>Disable status line</td>
</tr>
<tr>
<td>display_clock</td>
<td>dclk</td>
<td>DK</td>
<td>Display time-of-day clock</td>
</tr>
<tr>
<td>display_pc_char</td>
<td>dispC</td>
<td>S1</td>
<td>Display PC character</td>
</tr>
<tr>
<td>down_half_line</td>
<td>hd</td>
<td>hd</td>
<td>Half-line down (forward 1/2 linefeed)</td>
</tr>
<tr>
<td>ena_acs</td>
<td>enacs</td>
<td>eA</td>
<td>Enable alternate character set</td>
</tr>
<tr>
<td>end_bit_image_region</td>
<td>endbi</td>
<td>Yy</td>
<td>End a bit-image region</td>
</tr>
<tr>
<td>enter_alt_charset_mode</td>
<td>smacs</td>
<td>as</td>
<td>Start alternate character set</td>
</tr>
<tr>
<td>enter_am_mode</td>
<td>smam</td>
<td>SA</td>
<td>Turn on automatic margins</td>
</tr>
<tr>
<td>enter_blink_mode</td>
<td>blink</td>
<td>mb</td>
<td>Turn on blinking</td>
</tr>
<tr>
<td>enter_bold_mode</td>
<td>bold</td>
<td>md</td>
<td>Turn on bold (extra bright) mode</td>
</tr>
<tr>
<td>enter_ca_mode</td>
<td>smcup</td>
<td>ti</td>
<td>String to begin programs that use cup</td>
</tr>
<tr>
<td>enter_delete_mode</td>
<td>smdc</td>
<td>dm</td>
<td>Delete mode (enter)</td>
</tr>
<tr>
<td>enter_dim_mode</td>
<td>dim</td>
<td>mh</td>
<td>Turn on half-bright mode</td>
</tr>
<tr>
<td>enter_doublewide_mode</td>
<td>swidm</td>
<td>ZF</td>
<td>Enable double wide printing</td>
</tr>
<tr>
<td>enter_draft_quality</td>
<td>sdrfq</td>
<td>ZG</td>
<td>Set draft quality print</td>
</tr>
<tr>
<td>enter_horizontal_hl_mode</td>
<td>ehhlm</td>
<td></td>
<td>Turn on horizontal highlight mode</td>
</tr>
<tr>
<td>enter_insert_mode</td>
<td>smir</td>
<td>im</td>
<td>Insert mode (enter)</td>
</tr>
<tr>
<td>enter_italics_mode</td>
<td>sitm</td>
<td>ZH</td>
<td>Enable italics</td>
</tr>
<tr>
<td>enter_left_hl_mode</td>
<td>ehlm</td>
<td></td>
<td>Turn on left highlight mode</td>
</tr>
<tr>
<td>enter_leftward_mode</td>
<td>slm</td>
<td>ZI</td>
<td>Enable leftward carriage motion</td>
</tr>
<tr>
<td>Variable</td>
<td>Capname</td>
<td>Termcap</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------</td>
<td>---------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>enter_low_hl_mode</td>
<td>elohlm</td>
<td>ZJ</td>
<td>Turn on low highlight mode</td>
</tr>
<tr>
<td>enter_micro_mode</td>
<td>smicm</td>
<td>ZJ</td>
<td>Enable micro motion capabilities</td>
</tr>
<tr>
<td>enter_near_letter_quality</td>
<td>snlq</td>
<td>ZL</td>
<td>Set near-letter quality print</td>
</tr>
<tr>
<td>enter_normal_quality</td>
<td>srmq</td>
<td>ZL</td>
<td>Set normal quality print</td>
</tr>
<tr>
<td>enter_pc_charset_mode</td>
<td>smpch</td>
<td>Z2</td>
<td>Enter PC character display mode</td>
</tr>
<tr>
<td>enter_protected_mode</td>
<td>prot</td>
<td>mp</td>
<td>Turn on protected mode</td>
</tr>
<tr>
<td>enter_reverse_mode</td>
<td>rev</td>
<td>mr</td>
<td>Turn on reverse video mode</td>
</tr>
<tr>
<td>enter_right_hl_mode</td>
<td>erhlm</td>
<td>ZP</td>
<td>Turn on right highlight mode</td>
</tr>
<tr>
<td>enter_scancode_mode</td>
<td>smsc</td>
<td>S4</td>
<td>Enter PC scancode mode</td>
</tr>
<tr>
<td>enter_secure_mode</td>
<td>invis</td>
<td>ZK</td>
<td>Turn on blank mode (characters invisible)</td>
</tr>
<tr>
<td>enter_shadow_mode</td>
<td>sshm</td>
<td>ZM</td>
<td>Enable shadow printing</td>
</tr>
<tr>
<td>enter_standout_mode</td>
<td>smso</td>
<td>so</td>
<td>Begin standout mode</td>
</tr>
<tr>
<td>enter subscrip_mode</td>
<td>ssubm</td>
<td>ZN</td>
<td>Enable subscript printing</td>
</tr>
<tr>
<td>enter_superscript_mode</td>
<td>ssupm</td>
<td>ZO</td>
<td>Enable superscript printing</td>
</tr>
<tr>
<td>enter_top_hl_mode</td>
<td>ethlm</td>
<td>ZP</td>
<td>Turn on top highlight mode</td>
</tr>
<tr>
<td>enter_underline_mode</td>
<td>smul</td>
<td>us</td>
<td>Start underscore mode</td>
</tr>
<tr>
<td>enter_upward_mode</td>
<td>sum</td>
<td>ZP</td>
<td>Enable upward carriage motion</td>
</tr>
<tr>
<td>enter_vertical_hl_mode</td>
<td>evhlm</td>
<td>ZP</td>
<td>Turn on vertical highlight mode</td>
</tr>
<tr>
<td>enter_xon_mode</td>
<td>srxon</td>
<td>SX</td>
<td>Turn on xon/xoff handshaking</td>
</tr>
<tr>
<td>erase_chars</td>
<td>ech</td>
<td>ec</td>
<td>Erase #1 characters</td>
</tr>
<tr>
<td>exit_alt_charset_mode</td>
<td>rmacs</td>
<td>ae</td>
<td>End alternate character set</td>
</tr>
<tr>
<td>exit_am_mode</td>
<td>rmam</td>
<td>RA</td>
<td>Turn off automatic margins</td>
</tr>
<tr>
<td>exit_attribute_mode</td>
<td>sgr0</td>
<td>me</td>
<td>Turn off all attributes</td>
</tr>
<tr>
<td>exit_ca_mode</td>
<td>rmcup</td>
<td>te</td>
<td>String to end programs that use cup</td>
</tr>
<tr>
<td>exit_delete_mode</td>
<td>rmdc</td>
<td>ed</td>
<td>End delete mode</td>
</tr>
<tr>
<td>exit_doublewide_mode</td>
<td>rwidm</td>
<td>ZQ</td>
<td>Disable double wide printing</td>
</tr>
<tr>
<td>exit_insert_mode</td>
<td>rmir</td>
<td>e1</td>
<td>End insert mode</td>
</tr>
<tr>
<td>exit_italics_mode</td>
<td>ritm</td>
<td>ZR</td>
<td>Disable italics</td>
</tr>
<tr>
<td>exit_leftward_mode</td>
<td>rlm</td>
<td>ZS</td>
<td>Enable leftward (normal) carriage motion</td>
</tr>
<tr>
<td>exit_micro_mode</td>
<td>rmicm</td>
<td>ZT</td>
<td>Disable micro motion capabilities</td>
</tr>
<tr>
<td>exit_pc_charset_mode</td>
<td>rmpch</td>
<td>S3</td>
<td>Disable PC character display mode</td>
</tr>
<tr>
<td>exit_scancode_mode</td>
<td>rmse</td>
<td>S5</td>
<td>Disable PC scancode mode</td>
</tr>
<tr>
<td>exit_shadow_mode</td>
<td>rshm</td>
<td>ZU</td>
<td>Disable shadow printing</td>
</tr>
<tr>
<td>exit_standout_mode</td>
<td>rmso</td>
<td>se</td>
<td>End standout mode</td>
</tr>
<tr>
<td>exit_subscript_mode</td>
<td>rsbubm</td>
<td>ZV</td>
<td>Disable subscript printing</td>
</tr>
<tr>
<td>exit_superscript_mode</td>
<td>rsupm</td>
<td>ZW</td>
<td>Disable superscript printing</td>
</tr>
<tr>
<td>exit_underline_mode</td>
<td>rmul</td>
<td>ue</td>
<td>End underscore mode</td>
</tr>
<tr>
<td>exit_upward_mode</td>
<td>rum</td>
<td>ZX</td>
<td>Enable downward (normal) carriage motion</td>
</tr>
<tr>
<td>exit_xon_mode</td>
<td>rmxon</td>
<td>RX</td>
<td>Turn off xon/xoff handshaking</td>
</tr>
<tr>
<td>fixed pause</td>
<td>pause</td>
<td>PA</td>
<td>Pause for 2-3 seconds</td>
</tr>
<tr>
<td>flash_hook</td>
<td>hook</td>
<td>fh</td>
<td>Flash the switch hook</td>
</tr>
<tr>
<td>flash_screen</td>
<td>flash</td>
<td>vb</td>
<td>Visible bell (may move cursor)</td>
</tr>
<tr>
<td>form_feed</td>
<td>ff</td>
<td>ff</td>
<td>Hardcopy terminal page eject</td>
</tr>
<tr>
<td>from status line</td>
<td>fsl</td>
<td>fs</td>
<td>Return from status line</td>
</tr>
<tr>
<td>get_mouse</td>
<td>getm</td>
<td>Gm</td>
<td>Curses should get button events</td>
</tr>
<tr>
<td>goto window</td>
<td>wingo</td>
<td>WG</td>
<td>Go to window #1</td>
</tr>
<tr>
<td>hangup</td>
<td>hup</td>
<td>HU</td>
<td>Hang-up phone</td>
</tr>
<tr>
<td>init_1string</td>
<td>is1</td>
<td>i1</td>
<td>Terminal or printer initialization string</td>
</tr>
<tr>
<td>init_2string</td>
<td>is2</td>
<td>is</td>
<td>Terminal or printer initialization string</td>
</tr>
</tbody>
</table>
### Source File Syntax

<table>
<thead>
<tr>
<th>Variable</th>
<th>Capname</th>
<th>Termcap</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>init_3string</td>
<td>is3</td>
<td>i3</td>
<td>Terminal or printer initialization string</td>
</tr>
<tr>
<td>init_file</td>
<td>if</td>
<td>if</td>
<td>Name of initialization file</td>
</tr>
<tr>
<td>init_prog</td>
<td>isprog</td>
<td>IP</td>
<td>Path name of program for initialization</td>
</tr>
<tr>
<td>initialize_color</td>
<td>initc</td>
<td>IC</td>
<td>Set color #1 to RGB #2, #3, #4</td>
</tr>
<tr>
<td>initialize_pair</td>
<td>initp</td>
<td>Ip</td>
<td>Set color-pair #1 to fg #2, bg #3</td>
</tr>
<tr>
<td>insert_character</td>
<td>ich1</td>
<td>ic</td>
<td>Insert character</td>
</tr>
<tr>
<td>insert_line</td>
<td>il1</td>
<td>a1</td>
<td>Add new blank line</td>
</tr>
<tr>
<td>insert_padding</td>
<td>ip</td>
<td>ip</td>
<td>Insert pad after character inserted</td>
</tr>
</tbody>
</table>

**Note:** The “key_” strings are sent by specific keys. The “key_” descriptions include the macro, defined in `<curses.h>`, for the code returned by `getch()` when the key is pressed (see `getch()`).

```plaintext
key_a1          ka1     K1    upper left of keypad
key_a3          ka3     K3    upper right of keypad
key_b2          kb2     k2    center of keypad
key_backspace   kbs     kb    sent by backspace key
key_beg         kbeg    @1   sent by beg(inning) key
key_btab        kcbt    kB    sent by back-tab key
key_c1          kc1     K4    lower left of keypad
key_c3          kc3     K5    lower right of keypad
key_cancel      kcan    @2   sent by cancel key
key_catab       ktbce   ka    sent by clear-all-tabs key
key_clear       kclr    kC    sent by clear-screen or erase key
key_close       kclo    @3   sent by close key
key_command     kcmd    @4   sent by cmd (command) key
key_copy        kcpy    @5   sent by copy key
key_create      kcr    @6   sent by create key
key_ctab        kctab   kt    sent by clear-tab key
key_dc          kdc1    k0    sent by delete-character key
key_d1          kdl1    kL    sent by delete-line key
key_down         kcd1   kd    sent by terminal down-arrow key
key_eic         krmir   @0   sent by `rmir` or `smir` in insert mode
key_find         kfnd    @0   sent by find key
key_help         khlp    @1   sent by help key
key_home         khome   kh    sent by home key
key_ic           kich1   kI    sent by ins-char/enter ins-mode key
key_il           kil1    kA    sent by insert-line key
key_left         kcub1   kL    sent by terminal left-arrow key
key_l1           kll     kH    sent by home-down key
key_mark         krmrk   %2   sent by mark key
key_message      kmsg    %3   sent by message key
```
<table>
<thead>
<tr>
<th>Variable</th>
<th>Capname</th>
<th>Termcap</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key_mouse</td>
<td>kmous</td>
<td>Km</td>
<td>0631, Mouse event has occurred</td>
</tr>
<tr>
<td>key_move</td>
<td>kmov</td>
<td>%4</td>
<td>sent by move key</td>
</tr>
<tr>
<td>key_next</td>
<td>knxt</td>
<td>%5</td>
<td>sent by next-object key</td>
</tr>
<tr>
<td>key_npage</td>
<td>knp</td>
<td>%6</td>
<td>sent by next-page key</td>
</tr>
<tr>
<td>key_open</td>
<td>kopn</td>
<td>%7</td>
<td>sent by open key</td>
</tr>
<tr>
<td>key_options</td>
<td>kopt</td>
<td>%8</td>
<td>sent by options key</td>
</tr>
<tr>
<td>key_previous</td>
<td>kpp</td>
<td>%9</td>
<td>sent by previous-page key</td>
</tr>
<tr>
<td>key_print</td>
<td>kpnt</td>
<td>&amp;0</td>
<td>sent by print or copy key</td>
</tr>
<tr>
<td>key_redo</td>
<td>krdo</td>
<td>&amp;1</td>
<td>sent by redo key</td>
</tr>
<tr>
<td>key_reference</td>
<td>kref</td>
<td>&amp;2</td>
<td>sent by reference key</td>
</tr>
<tr>
<td>key_refresh</td>
<td>krfr</td>
<td>&amp;3</td>
<td>sent by refresh key</td>
</tr>
<tr>
<td>key_replace</td>
<td>krpl</td>
<td>&amp;4</td>
<td>sent by replace key</td>
</tr>
<tr>
<td>key_restart</td>
<td>kres</td>
<td>&amp;5</td>
<td>sent by restart key</td>
</tr>
<tr>
<td>key_resume</td>
<td>kres</td>
<td>&amp;6</td>
<td>sent by resume key</td>
</tr>
<tr>
<td>key_right</td>
<td>kcuf1</td>
<td>kR</td>
<td>sent by terminal right-arrow key</td>
</tr>
<tr>
<td>key_save</td>
<td>ksav</td>
<td>&amp;7</td>
<td>sent by save key</td>
</tr>
<tr>
<td>key_sbeg</td>
<td>kBEG</td>
<td>&amp;8</td>
<td>sent by shifted beginning key</td>
</tr>
<tr>
<td>key_scancel</td>
<td>kCAN</td>
<td>&amp;9</td>
<td>sent by shifted cancel key</td>
</tr>
<tr>
<td>key_command</td>
<td>kCMD</td>
<td>*1</td>
<td>sent by shifted command key</td>
</tr>
<tr>
<td>key_scopy</td>
<td>kCPY</td>
<td>*2</td>
<td>sent by shifted copy key</td>
</tr>
<tr>
<td>key_screate</td>
<td>kCRT</td>
<td>*3</td>
<td>sent by shifted create key</td>
</tr>
<tr>
<td>key_sdc</td>
<td>kDC</td>
<td>*4</td>
<td>sent by shifted delete-char key</td>
</tr>
<tr>
<td>key SDL</td>
<td>kDL</td>
<td>*5</td>
<td>sent by shifted delete-line key</td>
</tr>
<tr>
<td>key_select</td>
<td>kslt</td>
<td>*6</td>
<td>sent by select key</td>
</tr>
<tr>
<td>key_send</td>
<td>kEND</td>
<td>*7</td>
<td>sent by shifted end key</td>
</tr>
<tr>
<td>key_seol</td>
<td>kEOL</td>
<td>*8</td>
<td>sent by shifted clear-line key</td>
</tr>
<tr>
<td>key_sexit</td>
<td>kEXT</td>
<td>*9</td>
<td>sent by shifted exit key</td>
</tr>
<tr>
<td>key sf</td>
<td>kind</td>
<td>kF</td>
<td>sent by scroll-forward/down key</td>
</tr>
<tr>
<td>key_sfind</td>
<td>kFND</td>
<td>&amp;0</td>
<td>sent by shifted find key</td>
</tr>
<tr>
<td>key_shelp</td>
<td>kHLP</td>
<td>&amp;1</td>
<td>sent by shifted help key</td>
</tr>
<tr>
<td>key_shome</td>
<td>kHOM</td>
<td>&amp;2</td>
<td>sent by shifted home key</td>
</tr>
<tr>
<td>key_sic</td>
<td>kIC</td>
<td>&amp;3</td>
<td>sent by shifted input key</td>
</tr>
<tr>
<td>key_sleft</td>
<td>kLFT</td>
<td>&amp;4</td>
<td>sent by shifted left-arrow key</td>
</tr>
<tr>
<td>key_smessages</td>
<td>kMSG</td>
<td>&amp;5</td>
<td>sent by shifted message key</td>
</tr>
<tr>
<td>key_smoved</td>
<td>kMOV</td>
<td>&amp;6</td>
<td>sent by shifted move key</td>
</tr>
<tr>
<td>key_snext</td>
<td>kNXT</td>
<td>&amp;7</td>
<td>sent by shifted next key</td>
</tr>
<tr>
<td>key_soptions</td>
<td>kOPT</td>
<td>&amp;8</td>
<td>sent by shifted options key</td>
</tr>
<tr>
<td>key_sprevious</td>
<td>kPRV</td>
<td>&amp;9</td>
<td>sent by shifted prev key</td>
</tr>
<tr>
<td>key_sprint</td>
<td>kPRT</td>
<td>&amp;0</td>
<td>sent by shifted print key</td>
</tr>
<tr>
<td>key_sr</td>
<td>kri</td>
<td>kR</td>
<td>sent by scroll-backward/up key</td>
</tr>
<tr>
<td>key_sredo</td>
<td>kRDO</td>
<td>&amp;1</td>
<td>sent by shifted redo key</td>
</tr>
<tr>
<td>key_sreplace</td>
<td>kRPL</td>
<td>&amp;2</td>
<td>sent by shifted replace key</td>
</tr>
<tr>
<td>key_sright</td>
<td>kRIT</td>
<td>&amp;3</td>
<td>sent by shifted right-arrow key</td>
</tr>
<tr>
<td>key_sresume</td>
<td>kRES</td>
<td>&amp;4</td>
<td>sent by shifted resume key</td>
</tr>
<tr>
<td>key_ssave</td>
<td>kSAV</td>
<td>&amp;5</td>
<td>sent by shifted save key</td>
</tr>
<tr>
<td>key_ssuspend</td>
<td>kSPD</td>
<td>&amp;6</td>
<td>sent by shifted suspend key</td>
</tr>
<tr>
<td>keystab</td>
<td>kHts</td>
<td>&amp;7</td>
<td>sent by set-tab key</td>
</tr>
<tr>
<td>key_sunddo</td>
<td>kUND</td>
<td>&amp;8</td>
<td>sent by shifted undo key</td>
</tr>
<tr>
<td>key_suspend</td>
<td>kspd</td>
<td>&amp;9</td>
<td>sent by suspend key</td>
</tr>
<tr>
<td>key_undo</td>
<td>kund</td>
<td>#0</td>
<td>sent by undo key</td>
</tr>
<tr>
<td>key_up</td>
<td>kcuu1</td>
<td>ku</td>
<td>sent by terminal up-arrow key</td>
</tr>
<tr>
<td>keypad_local</td>
<td>rmkx</td>
<td>ke</td>
<td>Out of “keypad-transmit” mode</td>
</tr>
</tbody>
</table>
### Source File Syntax

<table>
<thead>
<tr>
<th>Variable</th>
<th>Capname</th>
<th>Termcap</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keypad_xmit</td>
<td>smkx</td>
<td>ks</td>
<td>Put terminal in “keypad-transmit” mode</td>
</tr>
<tr>
<td>lab_f0</td>
<td>lf0</td>
<td>10</td>
<td>Labels on function key f0 if not f0</td>
</tr>
<tr>
<td>lab_f1</td>
<td>lf1</td>
<td>11</td>
<td>Labels on function key f1 if not f1</td>
</tr>
<tr>
<td>lab_f2</td>
<td>lf2</td>
<td>12</td>
<td>Labels on function key f2 if not f2</td>
</tr>
<tr>
<td>lab_f3</td>
<td>lf3</td>
<td>13</td>
<td>Labels on function key f3 if not f3</td>
</tr>
<tr>
<td>lab_f4</td>
<td>lf4</td>
<td>14</td>
<td>Labels on function key f4 if not f4</td>
</tr>
<tr>
<td>lab_f5</td>
<td>lf5</td>
<td>15</td>
<td>Labels on function key f5 if not f5</td>
</tr>
<tr>
<td>lab_f6</td>
<td>lf6</td>
<td>16</td>
<td>Labels on function key f6 if not f6</td>
</tr>
<tr>
<td>lab_f7</td>
<td>lf7</td>
<td>17</td>
<td>Labels on function key f7 if not f7</td>
</tr>
<tr>
<td>lab_f8</td>
<td>lf8</td>
<td>18</td>
<td>Labels on function key f8 if not f8</td>
</tr>
<tr>
<td>lab_f9</td>
<td>lf9</td>
<td>19</td>
<td>Labels on function key f9 if not f9</td>
</tr>
<tr>
<td>lab_f10</td>
<td>lf10</td>
<td>1a</td>
<td>Labels on function key f10 if not f10</td>
</tr>
<tr>
<td>label_format</td>
<td>fln</td>
<td>Lf</td>
<td>Label format</td>
</tr>
<tr>
<td>label_off</td>
<td>rmln</td>
<td>LF</td>
<td>Turn off soft labels</td>
</tr>
<tr>
<td>label_on</td>
<td>smln</td>
<td>L0</td>
<td>Turn on soft labels</td>
</tr>
<tr>
<td>meta_off</td>
<td>rmm</td>
<td>mo</td>
<td>Turn off “meta mode”</td>
</tr>
<tr>
<td>meta_on</td>
<td>smm</td>
<td>mm</td>
<td>Turn on “meta mode” (8th bit)</td>
</tr>
<tr>
<td>micro_column_address</td>
<td>mhpa</td>
<td>ZY</td>
<td>Like column_address for micro adjustment</td>
</tr>
<tr>
<td>micro_down</td>
<td>mcud1</td>
<td>ZZ</td>
<td>Like cursor_down for micro adjustment</td>
</tr>
<tr>
<td>micro_left</td>
<td>mcub1</td>
<td>Za</td>
<td>Like cursor_left for micro adjustment</td>
</tr>
<tr>
<td>micro_right</td>
<td>mcuf1</td>
<td>Zb</td>
<td>Like cursor_right for micro adjustment</td>
</tr>
<tr>
<td>micro_row_address</td>
<td>mvpa</td>
<td>Zc</td>
<td>Like row_address for micro adjustment</td>
</tr>
<tr>
<td>micro_up</td>
<td>mcuu1</td>
<td>Zd</td>
<td>Like cursor_up for micro adjustment</td>
</tr>
<tr>
<td>mouse_info</td>
<td>minfo</td>
<td>Mi</td>
<td>Mouse status information</td>
</tr>
<tr>
<td>newline</td>
<td>nel</td>
<td>mw</td>
<td>Newline (behaves like cr followed by lf)</td>
</tr>
<tr>
<td>order_of_pins</td>
<td>porder</td>
<td>Ze</td>
<td>Matches software bits to print-head pins</td>
</tr>
<tr>
<td>orig_colors</td>
<td>oc</td>
<td>oc</td>
<td>Set all color(-pair)s to the original ones</td>
</tr>
<tr>
<td>orig_pair</td>
<td>op</td>
<td>op</td>
<td>Set default color-pair to the original one</td>
</tr>
<tr>
<td>pad_char</td>
<td>pad</td>
<td>pc</td>
<td>Pad character (rather than null)</td>
</tr>
<tr>
<td>parm_dch</td>
<td>dch</td>
<td>DC</td>
<td>Delete #1 chars</td>
</tr>
<tr>
<td>parm_delete_line</td>
<td>dl</td>
<td>DL</td>
<td>Delete #1 lines</td>
</tr>
<tr>
<td>parm_down_cursor</td>
<td>cud</td>
<td>DO</td>
<td>Move down #1 lines</td>
</tr>
<tr>
<td>parm_down_micro</td>
<td>mcud</td>
<td>Zf</td>
<td>Like parm_down_cursor for micro adjust.</td>
</tr>
<tr>
<td>parm_ich</td>
<td>ich</td>
<td>IC</td>
<td>Insert #1 blank chars</td>
</tr>
<tr>
<td>parm_index</td>
<td>indn</td>
<td>SF</td>
<td>Scroll forward #1 lines</td>
</tr>
<tr>
<td>parm_insert_line</td>
<td>il</td>
<td>AL</td>
<td>Add #1 new blank lines</td>
</tr>
<tr>
<td>parm_left_cursor</td>
<td>cub</td>
<td>LE</td>
<td>Move cursor left #1 spaces</td>
</tr>
<tr>
<td>parm_left_micro</td>
<td>mcub</td>
<td>Zg</td>
<td>Like parm_left_cursor for micro adjust.</td>
</tr>
<tr>
<td>parm_right_cursor</td>
<td>cuf</td>
<td>RI</td>
<td>Move right #1 spaces</td>
</tr>
<tr>
<td>parm_right_micro</td>
<td>mcuf</td>
<td>Zh</td>
<td>Like parm_right_cursor for micro adjust.</td>
</tr>
<tr>
<td>parm_rindex</td>
<td>rin</td>
<td>SR</td>
<td>Scroll backward #1 lines</td>
</tr>
<tr>
<td>Variable</td>
<td>Capname</td>
<td>Termcap</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>parm_up_cursor</td>
<td>cuu</td>
<td>UP</td>
<td>Move cursor up #1 lines.</td>
</tr>
<tr>
<td>parm_up_micro</td>
<td>mcuu</td>
<td>Z1</td>
<td>Like parm_up_cursor for micro adjust.</td>
</tr>
<tr>
<td>pc_term_options</td>
<td>pctrm</td>
<td>S6</td>
<td>PC terminal options</td>
</tr>
<tr>
<td>pkey_key</td>
<td>pfkey</td>
<td>pk</td>
<td>Prog funct key #1 to type string #2</td>
</tr>
<tr>
<td>pkey_local</td>
<td>pfloc</td>
<td>Pl</td>
<td>Prog funct key #1 to execute string #2</td>
</tr>
<tr>
<td>pkey_plab</td>
<td>pfxl</td>
<td>X1</td>
<td>Prog key #1 to xmit string #2 and show string #3</td>
</tr>
<tr>
<td>pkey_xmit</td>
<td>pfx</td>
<td>Px</td>
<td>Prog funct key #1 to xmit string #2</td>
</tr>
<tr>
<td>plab_norm</td>
<td>pln</td>
<td>Pn</td>
<td>Prog label #1 to show string #2</td>
</tr>
<tr>
<td>print_screen</td>
<td>mc0</td>
<td>Ps</td>
<td>Print contents of the screen</td>
</tr>
<tr>
<td>prtr_non</td>
<td>mc5p</td>
<td>P0</td>
<td>Turn on the printer for #1 bytes</td>
</tr>
<tr>
<td>prtr_off</td>
<td>mc4f</td>
<td>Pf</td>
<td>Turn off the printer</td>
</tr>
<tr>
<td>prtr_on</td>
<td>mc5p</td>
<td>P0</td>
<td>Turn on the printer</td>
</tr>
<tr>
<td>pulse</td>
<td>pulse</td>
<td>Pu</td>
<td>Select pulse dialing</td>
</tr>
<tr>
<td>quick_dial</td>
<td>qdial</td>
<td>Q0</td>
<td>Dial phone number #1, without progress detection</td>
</tr>
<tr>
<td>remove_clock</td>
<td>rmclkr</td>
<td>RC</td>
<td>Remove time-of-day clock</td>
</tr>
<tr>
<td>repeat_char</td>
<td>rep</td>
<td>Rp</td>
<td>Repeat char #1 #2 times</td>
</tr>
<tr>
<td>req_for_input</td>
<td>rfi</td>
<td>RF</td>
<td>Send next input char (for ptys)</td>
</tr>
<tr>
<td>req_mouse_pos</td>
<td>reqmp</td>
<td>R0</td>
<td>Request mouse position report</td>
</tr>
<tr>
<td>reset_1string</td>
<td>rs1</td>
<td>R1</td>
<td>Reset terminal completely to sane modes</td>
</tr>
<tr>
<td>reset_2string</td>
<td>rs2</td>
<td>R2</td>
<td>Reset terminal completely to sane modes</td>
</tr>
<tr>
<td>reset_3string</td>
<td>rs3</td>
<td>R3</td>
<td>Reset terminal completely to sane modes</td>
</tr>
<tr>
<td>reset_file</td>
<td>rf</td>
<td>Rf</td>
<td>Name of file containing reset string</td>
</tr>
<tr>
<td>restore_cursor</td>
<td>rc</td>
<td>Rc</td>
<td>Restore cursor to position of last sc</td>
</tr>
<tr>
<td>row_address</td>
<td>vpa</td>
<td>Cv</td>
<td>Set vertical position to absolute #1</td>
</tr>
<tr>
<td>save_cursor</td>
<td>sc</td>
<td>Sc</td>
<td>Save cursor position</td>
</tr>
<tr>
<td>scancode_escape</td>
<td>scesc</td>
<td>Sc</td>
<td>Escape for scancode emulation</td>
</tr>
<tr>
<td>scroll_forward</td>
<td>ind</td>
<td>Sf</td>
<td>Scroll text up</td>
</tr>
<tr>
<td>scroll_reverse</td>
<td>ri</td>
<td>Sr</td>
<td>Scroll text down</td>
</tr>
<tr>
<td>select_char_set</td>
<td>sscs</td>
<td>Zj</td>
<td>Select character set</td>
</tr>
<tr>
<td>set0_des_seq</td>
<td>s0ds</td>
<td>S0</td>
<td>Shift into codeset 0 (EUC set 0, ASCII)</td>
</tr>
<tr>
<td>set1_des_seq</td>
<td>s1ds</td>
<td>S1</td>
<td>Shift into codeset 1</td>
</tr>
<tr>
<td>set2_des_seq</td>
<td>s2ds</td>
<td>S2</td>
<td>Shift into codeset 2</td>
</tr>
<tr>
<td>set3_des_seq</td>
<td>s3ds</td>
<td>S3</td>
<td>Shift into codeset 3</td>
</tr>
<tr>
<td>set_a_attributes</td>
<td>sgr1</td>
<td>Sa</td>
<td>Define second set of video attributes #1-#6</td>
</tr>
<tr>
<td>set_a_background</td>
<td>setab</td>
<td>Ab</td>
<td>Set background color to #1 using ANSI escape</td>
</tr>
<tr>
<td>set_a_foreground</td>
<td>setaf</td>
<td>Af</td>
<td>Set foreground color to #1 using ANSI escape</td>
</tr>
<tr>
<td>set_attributes</td>
<td>sgr</td>
<td>Sa</td>
<td>Define first set of video attributes #1-#9</td>
</tr>
<tr>
<td>set_background</td>
<td>setb</td>
<td>Sb</td>
<td>Set background color to #1</td>
</tr>
<tr>
<td>set_bottom_margin</td>
<td>smgb</td>
<td>Zk</td>
<td>Set bottom margin at current line</td>
</tr>
<tr>
<td>set_bottom_margin_parm</td>
<td>smgbp</td>
<td>Zl</td>
<td>Set bottom margin at line #1 or #2 lines from bottom</td>
</tr>
<tr>
<td>set_clock</td>
<td>sclk</td>
<td>Sc</td>
<td>Set clock to hours (#1), minutes (#2), seconds (#3)</td>
</tr>
<tr>
<td>set_color_band</td>
<td>setcolor</td>
<td>Yz</td>
<td>Change to ribbon color #1</td>
</tr>
</tbody>
</table>
### Variable Capname Termcap Description

<table>
<thead>
<tr>
<th>Variable</th>
<th>Capname</th>
<th>Termcap</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_color_pair</td>
<td>scp</td>
<td>sp</td>
<td>Set current color pair to #1</td>
</tr>
<tr>
<td>set_foreground</td>
<td>setf</td>
<td>Sf</td>
<td>Set foreground color to #1</td>
</tr>
<tr>
<td>set_left_margin</td>
<td>smgl</td>
<td>ML</td>
<td>Set left margin at current column</td>
</tr>
<tr>
<td>set_left_margin_parm</td>
<td>smglp</td>
<td>Zm</td>
<td>Set left (right) margin at column #1 (#2)</td>
</tr>
<tr>
<td>set_lr_margin</td>
<td>smglr</td>
<td>ML</td>
<td>Sets both left and right margins</td>
</tr>
<tr>
<td>set_page_length</td>
<td>lines</td>
<td>YZ</td>
<td>Set page length to #1 lines</td>
</tr>
<tr>
<td>set_pglen_inch</td>
<td>length</td>
<td>YI</td>
<td>Set page length to #1 hundredth of an inch</td>
</tr>
<tr>
<td>set_right_margin</td>
<td>smgr</td>
<td>MR</td>
<td>Set right margin at current column</td>
</tr>
<tr>
<td>set_right_margin_parm</td>
<td>smgrp</td>
<td>Zn</td>
<td>Set right margin at column #1</td>
</tr>
<tr>
<td>set_right_margin_parm</td>
<td>smgtb</td>
<td>MT</td>
<td>Set a tab in all rows, current column</td>
</tr>
<tr>
<td>set_right_margin_parm</td>
<td>smgt</td>
<td>Zo</td>
<td>Set top margin at current line</td>
</tr>
<tr>
<td>set_right_margin_parm</td>
<td>smgtp</td>
<td>Zp</td>
<td>Set top (bottom) margin at line #1 (#2)</td>
</tr>
<tr>
<td>set_window</td>
<td>wind</td>
<td>wi</td>
<td>Current window is lines #1-#2 cols #3-#4</td>
</tr>
<tr>
<td>start_bit_image</td>
<td>sbim</td>
<td>Zq</td>
<td>Start printing bit image graphics</td>
</tr>
<tr>
<td>start_char_set_def</td>
<td>scsd</td>
<td>Zr</td>
<td>Start definition of a character set</td>
</tr>
<tr>
<td>stop_bit_image</td>
<td>rbim</td>
<td>Zs</td>
<td>End printing bit image graphics</td>
</tr>
<tr>
<td>stop_char_set_def</td>
<td>rcsd</td>
<td>Zt</td>
<td>End definition of a character set</td>
</tr>
<tr>
<td>subscript_characters</td>
<td>subcs</td>
<td>Zu</td>
<td>List of “subscript-able” characters</td>
</tr>
<tr>
<td>superscript_characters</td>
<td>supcs</td>
<td>Zv</td>
<td>List of “superscript-able” characters</td>
</tr>
<tr>
<td>tab</td>
<td>ht</td>
<td>ta</td>
<td>Tab to next 8-space hardware tab stop</td>
</tr>
<tr>
<td>these_cause_cr</td>
<td>docr</td>
<td>Zw</td>
<td>Printing any of these chars causes cr</td>
</tr>
<tr>
<td>to_status_line</td>
<td>tsl</td>
<td>ts</td>
<td>Go to status line, col #1</td>
</tr>
<tr>
<td>tone</td>
<td>tone</td>
<td>T0</td>
<td>Select touch tone dialing</td>
</tr>
<tr>
<td>user0</td>
<td>u0</td>
<td>u0</td>
<td>User string 0</td>
</tr>
<tr>
<td>user1</td>
<td>u1</td>
<td>u1</td>
<td>User string 1</td>
</tr>
<tr>
<td>user2</td>
<td>u2</td>
<td>u2</td>
<td>User string 2</td>
</tr>
<tr>
<td>user3</td>
<td>u3</td>
<td>u3</td>
<td>User string 3</td>
</tr>
<tr>
<td>user4</td>
<td>u4</td>
<td>u4</td>
<td>User string 4</td>
</tr>
<tr>
<td>user5</td>
<td>u5</td>
<td>u5</td>
<td>User string 5</td>
</tr>
<tr>
<td>user6</td>
<td>u6</td>
<td>u6</td>
<td>User string 6</td>
</tr>
<tr>
<td>user7</td>
<td>u7</td>
<td>u7</td>
<td>User string 7</td>
</tr>
<tr>
<td>user8</td>
<td>u8</td>
<td>u8</td>
<td>User string 8</td>
</tr>
<tr>
<td>user9</td>
<td>u9</td>
<td>u9</td>
<td>User string 9</td>
</tr>
<tr>
<td>underline_char</td>
<td>uc</td>
<td>uc</td>
<td>Underscore one char and move past it</td>
</tr>
<tr>
<td>up_half_line</td>
<td>hu</td>
<td>hu</td>
<td>Half-line up (reverse 1/2 linefeed)</td>
</tr>
<tr>
<td>wait_tone</td>
<td>wait</td>
<td>WA</td>
<td>Wait for dial tone</td>
</tr>
<tr>
<td>xoff_character</td>
<td>xoffc</td>
<td>XF</td>
<td>X-off character</td>
</tr>
<tr>
<td>xon_character</td>
<td>xonc</td>
<td>XN</td>
<td>X-on character</td>
</tr>
<tr>
<td>zero_motion</td>
<td>zerom</td>
<td>Zx</td>
<td>No motion for the subsequent character</td>
</tr>
</tbody>
</table>

### Sample Entry

The following entry describes the AT&T; 610 terminal.
Types of Capabilities in the Sample Entry

The sample entry shows the formats for the three types of terminfo capabilities: 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 Capname column of the previous tables).

Boolean Capabilities

A boolean capability is true if its Capname is present in the entry, and false if its Capname is not present in the entry.

The ‘@’ character following a Capname is used to explicitly declare that a boolean capability is false.

Numeric Capabilities

Numeric capabilities are followed by the character '#' and then a positive integer value. The example assigns the value 80 to the cols numeric capability by coding: cols#80

Values for numeric capabilities may be specified in decimal, octal or hexadecimal, using normal C-language conventions.

String Capabilities

String-valued capabilities such as el (clear to end of line sequence) are listed by the 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 $el=\EK<3>$. The Curses implementation achieves delays by outputting to the terminal an appropriate number of system-defined padding characters. The $tputs()$ function provides delays when used to send such a capability to the terminal.

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 required delay is proportional to the number of lines affected by the operation, and the amount given is the delay required per affected unit. (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 delay is mandatory and padding characters are transmitted regardless of the setting of $xon$. If ‘/’ is not specified or if a device has $xon$ defined, the delay information is advisory and is only used for cost estimates or when the device is in raw mode. However, any delay specified for $bel$ or $flash$ is treated as mandatory.

The following notation is valid in terminfo source files for specifying special characters:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Represents Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>^x</td>
<td>Control-x (for any appropriate x)</td>
</tr>
<tr>
<td>\a</td>
<td>Alert</td>
</tr>
<tr>
<td>\b</td>
<td>Backspace</td>
</tr>
<tr>
<td>\E or \e</td>
<td>An ESCAPE character</td>
</tr>
<tr>
<td>\f</td>
<td>Form feed</td>
</tr>
<tr>
<td>\l</td>
<td>Linefeed</td>
</tr>
<tr>
<td>\n</td>
<td>Newline</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage return</td>
</tr>
<tr>
<td>\s</td>
<td>Space</td>
</tr>
<tr>
<td>\t</td>
<td>Tab</td>
</tr>
<tr>
<td>^</td>
<td>Caret (^)</td>
</tr>
<tr>
<td>\</td>
<td>Backslash ()</td>
</tr>
<tr>
<td>,</td>
<td>Comma (,)</td>
</tr>
<tr>
<td>:</td>
<td>Colon (:)</td>
</tr>
<tr>
<td>\0</td>
<td>Null</td>
</tr>
<tr>
<td>\nnn</td>
<td>Any character, specified as three octal digits</td>
</tr>
</tbody>
</table>

(See the XBD specification, General Terminal Interface.)

Commented-out Capabilities

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the second ind Note that capabilities are defined in a left-to-right order and, therefore, a prior definition will override a later definition.
Device Capabilities

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-xff 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 argument an scroll the number of lines specified by that argument.

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 if 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:
Parameterized Strings

Cursor addressing and other strings requiring arguments are described by a argumentized string capability with escapes in a form (%x) comparable to printf(). For example, to address the cursor, the cup capability is given, using two arguments: 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 argument mechanism uses a stack and special % codes to manipulate the stack in the manner of Reverse Polish Notation (postfix). Typically a sequence pushes one of the arguments onto the stack and then prints 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 argument, 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 the ith argument.

%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 integer2 op pop integer1) where integer1 represents the top of the stack.
Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen) then this can be given as home; similarly a fast way of getting to the lower left-hand corner can be given as ll; this may involve going up with cuu1 from the home position, but a program should never do this itself (unless ll does) because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as addressing to (0,0): to the top left
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 argument capabilities hpa (horizontal position absolute) and vpa (vertical position absolute). Sometimes these are shorter than the more general two-argument sequence (as with the Hewlett-Packard 2645) and can be used in preference to cup. If there are argumentized local motions (such as “move n spaces to the right”), these can be given as cud, cub, cuf, and cuu with a single argument 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 rmcup sequence will not restore the screen after an smcup sequence is output (to the state prior to outputting smcup), specify nrrmc.

Area Clears

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

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 argument 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 arguments: 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 argumentized string `wind`. The four arguments 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.

**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 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 argument, `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 \texttt{mir} to speed up inserting in this case. Omitting \texttt{mir} will affect only speed. Some terminals (notably Datamedia) must not have \texttt{mir} because of the way their insert mode works.

Finally, you can specify \texttt{dch1} to delete a single character, \texttt{dch} with one argument, \texttt{n}, to delete \texttt{n} characters, and delete mode by giving \texttt{smdc} and \texttt{rmdc} to enter and exit delete mode (any mode the terminal needs to be placed in for \texttt{dch1} to work).

A command to erase \texttt{n} characters (equivalent to outputting \texttt{n} blanks without moving the cursor) can be given as \texttt{ech} with one argument.

**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 (\texttt{blink})
- Bold or extra-bright characters (\texttt{bold})
- Dim or half-bright characters (\texttt{dim})
- Blanking or invisible text (\texttt{invis})
- Protected text (\texttt{prot})
- A reverse-video screen (\texttt{rev})
- An alternate character set (\texttt{smacs} to enter this mode and \texttt{rmacs} to exit it) (If a command is necessary before you can enter alternate character set mode, give the sequence in \texttt{enacs} or “enable alternate-character-set” mode.) Turning on any of these modes singly may turn off other modes.

\texttt{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 \texttt{dim} or \texttt{blink}.

Choose one display method as \texttt{standout mode} and use it to highlight error messages and other 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 \texttt{smso} and \texttt{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 \texttt{xmc} should be given to tell how many spaces are left.

Sequences to begin underlining and end underlining can be specified as \texttt{smul} and \texttt{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 \texttt{uc}.

Terminals with the “magic cookie” glitch (\texttt{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 \texttt{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 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 arguments. Each argument is either 0 or non-zero, as the corresponding attribute is on or off. The nine arguments 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.

<table>
<thead>
<tr>
<th><code>tparm</code> Argument</th>
<th>Attribute</th>
<th>Escape Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>standout</td>
<td>\E[0m</td>
</tr>
<tr>
<td>p1</td>
<td>underline</td>
<td>\E[0;4;7m</td>
</tr>
<tr>
<td>p2</td>
<td>reverse</td>
<td>\E[0;3m</td>
</tr>
<tr>
<td>p3</td>
<td>blink</td>
<td>\E[0;5m</td>
</tr>
<tr>
<td>p4</td>
<td>dim</td>
<td>\E[0;7m</td>
</tr>
<tr>
<td>p5</td>
<td>bold</td>
<td>\E[0;3;4m</td>
</tr>
<tr>
<td>p6</td>
<td>invis</td>
<td>\E[0;8m</td>
</tr>
<tr>
<td>p7</td>
<td>protect</td>
<td>not available</td>
</tr>
<tr>
<td>p8</td>
<td>altcharset</td>
<td>^O (off) ^N (on)</td>
</tr>
</tbody>
</table>

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, `p3` 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:
### 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 \texttt{smkx} and \texttt{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 \texttt{kcbu1}, \texttt{kcufl}, \texttt{kcuu1}, \texttt{kculd} and \texttt{kh}, respectively. If there are function keys such as \texttt{f0}, \texttt{f1}, ..., \texttt{f63}, the sequences they send can be specified as \texttt{kf0}, \texttt{kf1}, ..., \texttt{kf63}. If the first 11 keys have labels other than the default \texttt{f0} through \texttt{f10}, the labels can be given as \texttt{lf0}, \texttt{lf1}, ..., \texttt{lfl0}.

The codes transmitted by certain other special keys can be given: \texttt{kll} (home down), \texttt{kbs} (backspace), \texttt{ktbc} (clear all tabs), \texttt{kctab} (clear the tab stop in this column), \texttt{kclr} (clear screen or erase key), \texttt{kdc} (delete character), \texttt{kd} (delete line), \texttt{krmir} (exit insert mode), \texttt{kel} (clear to end of line), \texttt{k} (clear to end of screen), \texttt{kch} (insert character or enter insert mode), \texttt{kil} (insert line), \texttt{knp} (next page), \texttt{kpp} (previous page), \texttt{kind} (scroll forward/down), \texttt{kri} (scroll backward/up), \texttt{kh} (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 \texttt{ka1}, \texttt{ka3}, \texttt{kb2}, \texttt{kc1}, and \texttt{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 \texttt{pfkey}, \texttt{pfloc}, and \texttt{pfx}. A string to program screen labels should be specified as \texttt{pln}. Each of these strings takes two arguments: a function key identifier and a string to program it with. \texttt{pfkey} causes pressing the given key to be the same as the user typing the given string; \texttt{pfloc} causes the string to be executed by the terminal in local mode; and \texttt{pfx} causes the string to be transmitted to the computer. The capabilities \texttt{nlab}, \texttt{lw} and \texttt{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 \texttt{smln} and \texttt{rmln}. \texttt{smln} is normally output after one or more \texttt{pln} sequences to make sure that the change becomes visible.
Tabs and Initialization

If the device has hardware tabs, the command to advance to the next tab stop can be given as \texttt{ht} (usually control-I). A “backtab” command that moves leftward to the next tab stop can be given as \texttt{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 \texttt{ht} or \texttt{cbt} (even if they are present) because the user might 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 argument it is given, showing the number of spaces the tabs are set to. This is normally used by \texttt{tput init} 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 \texttt{terminfo} description can assume that they are properly set. If there are commands to set and clear tab stops, they can be given as \texttt{tbc} (clear all tab stops) and \texttt{hts} (set a tab stop in the current column of every row).

Other capabilities include: \texttt{is1}, \texttt{is2}, and \texttt{is3}, initialization strings for the device; \texttt{iprogs}, the path name of a program to be run to initialize the device; and \texttt{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 \texttt{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 \texttt{iprogs}; output \texttt{is1}; output \texttt{is2}; set the margins using \texttt{mgc}, \texttt{smgl} and \texttt{smgr}; set the tabs using \texttt{tbc} and \texttt{hts}; print the file if; and finally output \texttt{is3}. This is usually done using the \texttt{init} option of \texttt{tput}.

Most initialization is done with \texttt{is2}. Special device modes can be set up without duplicating strings by putting the common sequences in \texttt{is2} and special cases in \texttt{is1} and \texttt{is3}. Sequences that do a reset from a totally unknown state can be given as \texttt{rs1}, \texttt{rs2}, \texttt{rf}, and \texttt{rs3}, analogous to \texttt{is1}, \texttt{is2}, \texttt{is3}, and \texttt{if}. (The method using files, \texttt{if} and \texttt{rf}, is used for a few terminals however, the recommended method is to use the initialization and reset strings.) These strings are output by \texttt{tput reset}, which is used when the terminal gets into a wedged state. Commands are normally placed in \texttt{rs1}, \texttt{rs2}, \texttt{rs3}, and \texttt{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 \texttt{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 \texttt{tbc} and \texttt{hts}, the sequence can be placed in \texttt{is2} or \texttt{if}.

Any margin can be cleared with \texttt{mgc}. (For instructions on how to specify commands to set and clear margins.

Delays

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

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 argument, 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 (that is, `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 argument `wsl`.

### 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 Digital VT100 terminal, extended slightly with some characters from the AT&T; 4410v1 terminal.

<table>
<thead>
<tr>
<th>Glyph Name</th>
<th>VT100+ Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrow pointing right</td>
<td>+</td>
</tr>
<tr>
<td>arrow pointing left</td>
<td>,</td>
</tr>
<tr>
<td>arrow pointing down</td>
<td>.</td>
</tr>
<tr>
<td>solid square block</td>
<td>0</td>
</tr>
<tr>
<td>lantern symbol</td>
<td>I</td>
</tr>
<tr>
<td>arrow pointing up</td>
<td>-</td>
</tr>
<tr>
<td>diamond</td>
<td>`</td>
</tr>
<tr>
<td>checker board (stipple)</td>
<td>a</td>
</tr>
<tr>
<td>degree symbol</td>
<td>f</td>
</tr>
<tr>
<td>plus/minus</td>
<td>g</td>
</tr>
<tr>
<td>board of squares</td>
<td>h</td>
</tr>
<tr>
<td>lower right corner</td>
<td>j</td>
</tr>
<tr>
<td>upper right corner</td>
<td>k</td>
</tr>
<tr>
<td>upper left corner</td>
<td>l</td>
</tr>
<tr>
<td>lower left corner</td>
<td>m</td>
</tr>
<tr>
<td>plus</td>
<td>n</td>
</tr>
<tr>
<td>scan line 1</td>
<td>o</td>
</tr>
<tr>
<td>horizontal line</td>
<td>q</td>
</tr>
<tr>
<td>scan line 9</td>
<td>s</td>
</tr>
<tr>
<td>left tee (</td>
<td>-)</td>
</tr>
<tr>
<td>right tee (-</td>
<td>)</td>
</tr>
<tr>
<td>bottom tee (</td>
<td>)</td>
</tr>
<tr>
<td>top tee (</td>
<td>)</td>
</tr>
<tr>
<td>vertical line</td>
<td>x</td>
</tr>
<tr>
<td>bullet</td>
<td>˜</td>
</tr>
</tbody>
</table>

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 alternate-character-set mode. For example:
Glyph Name | VT100+ Character | Character Used on New Device
---|---|---
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; for example:
acsc=1RmFkTjGq\,x.

In addition, terminfo lets you define multiple character sets.

**Color Manipulation**

Most color terminals belong to one of two classes of terminal:

**Tektronix-style**

The Tektronix method uses a set of N predefined colors (usually 8) from which an application 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.

**Hewlett-Packard-style**

In the HP method, the application cannot define the foreground independently of the background, or vice-versa. Instead, the application must define an entire color-pair at once. Up to M color-pairs, made from 2*M different colors, can be defined this way.

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 *init_color()* functions 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.

If the terminal supports ANSI escape sequences to set background and foreground, they should be coded as *setab* and *setaf*, respectively. If the terminal supports other escape sequences to set background and foreground, they should be coded as *setb* and *setf*, respectively. The *vidputs()* function and the refresh functions use *setab* and *setaf* if they are defined. Each of these capabilities requires one argument: the
number of the color. By convention, the first eight colors (0-7) map to, in order: black, red, green, yellow, blue, magenta, cyan, white. However, color re-mapping may occur or the underlying hardware may not support these colors. Mappings for any additional colors supported by the device (that is, to numbers greater than 7) are at the discretion of the `terminfo` entry writer.

To initialize a color-pair (HP method), use `initp` (initialize pair). It requires seven arguments: 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 argument, 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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Bit Position</th>
<th>Decimal Value</th>
<th>Characteristic That Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA_ STANDOUT</td>
<td>0</td>
<td>1</td>
<td><code>sgr</code>, parameter 1</td>
</tr>
<tr>
<td>WA_ UNDERLINE</td>
<td>1</td>
<td>2</td>
<td><code>sgr</code>, parameter 2</td>
</tr>
<tr>
<td>WA_ REVERSE</td>
<td>2</td>
<td>4</td>
<td><code>sgr</code>, parameter 3</td>
</tr>
<tr>
<td>WA_ BLINK</td>
<td>3</td>
<td>8</td>
<td><code>sgr</code>, parameter 4</td>
</tr>
<tr>
<td>WA_ DIM</td>
<td>4</td>
<td>16</td>
<td><code>sgr</code>, parameter 5</td>
</tr>
<tr>
<td>WA_ BOLD</td>
<td>5</td>
<td>32</td>
<td><code>sgr</code>, parameter 6</td>
</tr>
<tr>
<td>WA_INVIS</td>
<td>6</td>
<td>64</td>
<td><code>sgr</code>, parameter 7</td>
</tr>
<tr>
<td>WA_PROTECT</td>
<td>7</td>
<td>128</td>
<td><code>sgr</code>, parameter 8</td>
</tr>
<tr>
<td>WA_ALTCAPSET</td>
<td>8</td>
<td>256</td>
<td><code>sgr</code>, parameter 9</td>
</tr>
<tr>
<td>WA_HORIZONTAL</td>
<td>9</td>
<td>512</td>
<td><code>sgr1</code>, parameter 1</td>
</tr>
<tr>
<td>WA_LEFT</td>
<td>10</td>
<td>1024</td>
<td><code>sgr1</code>, parameter 2</td>
</tr>
<tr>
<td>WA_LOW</td>
<td>11</td>
<td>2048</td>
<td><code>sgr1</code>, parameter 3</td>
</tr>
<tr>
<td>WA_RIGHT</td>
<td>12</td>
<td>4096</td>
<td><code>sgr1</code>, parameter 4</td>
</tr>
<tr>
<td>WA_TOP</td>
<td>13</td>
<td>8192</td>
<td><code>sgr1</code>, parameter 5</td>
</tr>
<tr>
<td>WA_VERTICAL</td>
<td>14</td>
<td>16384</td>
<td><code>sgr1</code>, parameter 6</td>
</tr>
</tbody>
</table>

When a particular video attribute should not be used with colors, set the corresponding `ncv` bit to 1; otherwise set it to 0. To determine the information to pack into the `ncv` variable, add 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).
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 argumentized string `rep`. The first argument 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 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 functions 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
- `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 argument, and leaves the printer on for as many characters as the value of the argument, then turns the printer off. The argument should not exceed 255. If the text is not displayed on the terminal screen when the printer is on, specify `mcSi` (silent printer). All text, including `mc4`, is transparently passed to the printer while an `mc5p` is in effect.

**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.

For Beehive Superbee terminals that do not transmit the escape or control-C characters, specify `xsbl`, indicating that the `f1` key is to be used for escape and the `f2` key for control-C.

**Similar Terminals**

If there are two 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 `capability-name@` prior to the appearance of the string capability use. For example, the entry:

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

defines an AT&T; 04424 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.

**Printer Capabilities**

The `terminfo` database lets you define capabilities of printers as well as terminals.
Rounding Values

Because argumentized string capabilities work only with integer values, terminfo designers should 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 argumentized string capability.

Printer Resolution

A printer’s resolution is defined to be the smallest spacing of characters it can achieve. In general, the horizontal and vertical resolutions are independent. 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 leftmost 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.

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:

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Number of Smallest Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>orhi</td>
<td>Steps per inch horizontally</td>
</tr>
<tr>
<td>orvi</td>
<td>Steps per inch vertically</td>
</tr>
<tr>
<td>orc</td>
<td>Steps per column</td>
</tr>
<tr>
<td>orl</td>
<td>Steps per line</td>
</tr>
</tbody>
</table>

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.

**Automatic Motion after Printing**

*Normal Mode:*

<table>
<thead>
<tr>
<th>orc</th>
<th>Steps moved horizontally</th>
</tr>
</thead>
<tbody>
<tr>
<td>orl</td>
<td>Steps moved vertically</td>
</tr>
</tbody>
</table>

*Micro Mode:*

<table>
<thead>
<tr>
<th>mcs</th>
<th>Steps moved horizontally</th>
</tr>
</thead>
<tbody>
<tr>
<td>mls</td>
<td>Steps moved vertically</td>
</tr>
</tbody>
</table>

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.

**Automatic Motion after Printing Wide Character**

*Normal Mode or Micro Mode (\(mcs = orc\)):*

| wides  | 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:

<table>
<thead>
<tr>
<th>Changing the Character/Line Pitches</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cpi</td>
<td>Change character pitch</td>
</tr>
<tr>
<td>cpix</td>
<td>If set, cpi changes orhi, otherwise changes orc</td>
</tr>
<tr>
<td>lpi</td>
<td>Change line pitch</td>
</tr>
<tr>
<td>lpx</td>
<td>If set, lpi changes orvi, otherwise changes orl</td>
</tr>
<tr>
<td>chr</td>
<td>Change steps per column</td>
</tr>
<tr>
<td>cvr</td>
<td>Change steps per line</td>
</tr>
</tbody>
</table>

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.

Programs that use **cpi**, **lpi**, **chr**, or **cvr** should recalculate the printer resolution (and
should recalculate other values).

### 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.

<table>
<thead>
<tr>
<th>String Capabilities for Motion</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mcub1</td>
<td>Move 1 step left</td>
</tr>
<tr>
<td>mcuf1</td>
<td>Move 1 step right</td>
</tr>
<tr>
<td>mcuu1</td>
<td>Move 1 step up</td>
</tr>
<tr>
<td>mcud1</td>
<td>Move 1 step down</td>
</tr>
<tr>
<td>mcub</td>
<td>Move N steps left</td>
</tr>
<tr>
<td>mcuf</td>
<td>Move N steps right</td>
</tr>
<tr>
<td>mcuu</td>
<td>Move N steps up</td>
</tr>
<tr>
<td>mcud</td>
<td>Move N steps down</td>
</tr>
<tr>
<td>mhpa</td>
<td>Move N steps from the left</td>
</tr>
<tr>
<td>mvpa</td>
<td>Move N steps from the top</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Limits to Motion</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mjump</td>
<td>Limit on use of <strong>mcub1</strong>, <strong>mcuf1</strong>, <strong>mcuu1</strong>, <strong>mcud1</strong></td>
</tr>
<tr>
<td>maddr</td>
<td>Limit on use of <strong>mhpa</strong>, <strong>mvpa</strong></td>
</tr>
<tr>
<td>xhpa</td>
<td>If set, <strong>hpa</strong> and <strong>mhpa</strong> can’t move left</td>
</tr>
<tr>
<td>xvpa</td>
<td>If set, <strong>vpa</strong> and <strong>mvpa</strong> can’t move up</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Entering/Exiting Micro Mode</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>smicm</td>
<td>Enter micro mode</td>
</tr>
<tr>
<td>rmicm</td>
<td>Exit micro mode</td>
</tr>
<tr>
<td>crxm</td>
<td>Using cr exits micro mode</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>What Happens After Character Printed in Rightmost Position</th>
<th>sam</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Automatic move to beginning of same line</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Entering/Exiting Reverse Modes</th>
<th>slm</th>
<th>rlm</th>
<th>sum</th>
<th>rum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reverse sense of horizontal motions</td>
<td>Restore sense of horizontal motions</td>
<td>Reverse sense of vertical motions</td>
<td>Restore sense of vertical motions</td>
</tr>
</tbody>
</table>

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 |
| cu1   | Move 1 column left |
| cub   | Move N columns right |
| cu    | 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 |
| cu1   | Move 1 line up |
| cuu   | Move N lines down |
| cu    | 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 rightmost position in 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 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.

<table>
<thead>
<tr>
<th>Miscellaneous Motion Strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>docr</td>
</tr>
<tr>
<td>zerom</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Setting Margins</th>
</tr>
</thead>
<tbody>
<tr>
<td>smgl</td>
</tr>
<tr>
<td>smgr</td>
</tr>
<tr>
<td>smgb</td>
</tr>
<tr>
<td>smgt</td>
</tr>
<tr>
<td>smgbp</td>
</tr>
<tr>
<td>smglp</td>
</tr>
<tr>
<td>smgrp</td>
</tr>
<tr>
<td>smgtp</td>
</tr>
</tbody>
</table>

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 argument 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 \texttt{smgp} and \texttt{smgrp} or \texttt{smgtp} and \texttt{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 \texttt{smgbp} means the bottom line of the page.

All margins can be cleared with \texttt{mgc}.

**Shadows, Italics, Wide Characters, Superscripts, Subscripts**

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

<table>
<thead>
<tr>
<th>Enhanced Printing</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sshm</td>
<td>Enter shadow-printing mode</td>
</tr>
<tr>
<td>rshm</td>
<td>Exit shadow-printing mode</td>
</tr>
<tr>
<td>sitm</td>
<td>Enter italicizing mode</td>
</tr>
<tr>
<td>ritm</td>
<td>Exit italicizing mode</td>
</tr>
<tr>
<td>swidm</td>
<td>Enter wide character mode</td>
</tr>
<tr>
<td>rwidm</td>
<td>Exit wide character mode</td>
</tr>
<tr>
<td>ssupm</td>
<td>Enter superscript mode</td>
</tr>
<tr>
<td>rsupm</td>
<td>Exit superscript mode</td>
</tr>
<tr>
<td>supcs</td>
<td>List of characters available as superscripts</td>
</tr>
<tr>
<td>ssubm</td>
<td>Enter subscript mode</td>
</tr>
<tr>
<td>rsubm</td>
<td>Exit subscript mode</td>
</tr>
<tr>
<td>subcs</td>
<td>List of characters available as subscripts</td>
</tr>
</tbody>
</table>

If a printer requires the \texttt{sshm} control sequence before every character to be shadow-printed, the \texttt{rshm} string is left blank. Thus programs that find a control sequence in \texttt{sshm} but none in \texttt{rshm} should use the \texttt{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 \texttt{rshm}. The same is also true of each of the \texttt{sitm/ritm}, \texttt{swidm/rwidm}, \texttt{ssupm/rsupm}, and \texttt{ssubm/rsubm} pairs.

\texttt{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 \texttt{widcs}.

If only a subset of the printable ASCII characters can be printed as superscripts or subscripts, they should be listed in \texttt{supcs} or \texttt{subcs} strings, respectively. If the
ssupm or ssunm 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. 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.

Alternate Character Sets

In addition to allowing you to define line graphics, terminfo lets you define alternate character sets. The following capabilities cover printers and terminals with multiple selectable or definable character sets:

<table>
<thead>
<tr>
<th>Alternate Character Sets</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scs</td>
<td>Select character set N</td>
</tr>
<tr>
<td>scsd</td>
<td>Start definition of character set N, M characters</td>
</tr>
<tr>
<td>defc</td>
<td>Define character A, B dots wide, descender D</td>
</tr>
<tr>
<td>rcsd</td>
<td>End definition of character set N</td>
</tr>
<tr>
<td>csnm</td>
<td>List of character set names</td>
</tr>
<tr>
<td>daisy</td>
<td>Printer has manually changed print-wheels</td>
</tr>
</tbody>
</table>

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 pointer to be returned by 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 pointer to be returned by 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.

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.

**Dot-Matrix Graphics**

Dot-matrix printers typically have the capability of reproducing raster graphics images. Three numeric capabilities and three string capabilities 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.

<table>
<thead>
<tr>
<th>Dot-Matrix Graphics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>npins</td>
<td>Number of pins, $N$, in print-head</td>
</tr>
<tr>
<td>spinv</td>
<td>Spacing of pins vertically in pins per inch</td>
</tr>
<tr>
<td>spinh</td>
<td>Spacing of dots horizontally in dots per inch</td>
</tr>
<tr>
<td>porder</td>
<td>Matches software bits to print-head pins</td>
</tr>
<tr>
<td>sbim</td>
<td>Start printing bit image graphics, $B$ bits wide</td>
</tr>
<tr>
<td>rbim</td>
<td>End printing bit image graphics</td>
</tr>
</tbody>
</table>

The sbim string 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 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 an 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, low-order bits of the last byte are set to 0. 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, Digital LA100, and Digital 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 `0,0,6,5,4,3,2,1;63`.

**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:

<table>
<thead>
<tr>
<th>Changing the Character/Line Pitches</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cpi</strong></td>
<td>Change character pitch</td>
</tr>
<tr>
<td><strong>cpix</strong></td>
<td>If set, cpi changes spinh</td>
</tr>
<tr>
<td><strong>lpi</strong></td>
<td>Change line pitch</td>
</tr>
<tr>
<td><strong>lpix</strong></td>
<td>If set, lpi changes spinv</td>
</tr>
</tbody>
</table>

`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.
Print Quality

Many dot-matrix printers can alter the dot spacing of printed text to produce near-letter-quality printing or draft-quality printing. It is important to be able to choose one or the other because the rate of printing generally decreases as the quality improves. Three strings describe these capabilities:

<table>
<thead>
<tr>
<th>Print Quality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>snlq</td>
<td>Set near-letter quality print</td>
</tr>
<tr>
<td>snrmq</td>
<td>Set normal quality print</td>
</tr>
<tr>
<td>sdrfq</td>
<td>Set draft quality print</td>
</tr>
</tbody>
</table>

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

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 numeric capabilities can help a program estimate what has been printed.

<table>
<thead>
<tr>
<th>Print Rate/Buffer Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cps</td>
<td>Nominal print rate in characters per second</td>
</tr>
<tr>
<td>bufsz</td>
<td>Buffer capacity in characters</td>
</tr>
</tbody>
</table>

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.

Selecting a Terminal

If the environment variable TERMINFO is defined, any program using Curses checks for a local terminal definition before checking in the standard place. For example, if TERM is set to att4424, then the compiled terminal definition is found in by default the path

a/att4424

within an implementation-specific directory.

(The a is copied from the first letter of att4424 to avoid creation of huge directories.) However, if TERMINFO is set to $HOME/myterms, Curses first checks

$HOME/myterms/a/att4424

If that fails, it then checks the default pathname.

This is useful for developing experimental definitions or when write permission in the implementation-defined default database is not available.

If the LINES and COLUMNS environment variables are set, or if the program is executing in a window environment, line and column information in the environment will override information read by terminfo.

Application Usage

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, 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 the terminfo database.

Conventions for Device Aliases

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:

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-w</td>
<td>Wide mode (more than 80 columns)</td>
<td>5410-w</td>
</tr>
<tr>
<td>-am</td>
<td>With automatic margins (usually default)</td>
<td>vt100-am</td>
</tr>
</tbody>
</table>
Variations of Terminal Definitions

It is implementation-defined how the entries in `terminfo` may be created.

There is more than one way to write a `terminfo` entry. A minimal entry may permit applications to use Curses to operate the terminal. If the entry is enhanced to describe more of the terminal’s capabilities, applications can use Curses to invoke those features, and can take advantages of optimizations within Curses and thus operate more efficiently. For most terminals, an optimal `terminfo` entry has already been written.
Source File Syntax
Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel
IBM Corporation
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

COPYRIGHT LICENSE:

This information might contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS™, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted
for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: [IBM Lifecycle Support for z/OS (http://www.ibm.com/software/support/systemsz/lifecycle/)]
- For information about currently-supported IBM hardware, contact your IBM representative.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml)

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, its affiliates, or both.
Glossary

background
A property of a window that specifies a character (the background character) and a rendition to be used in a variety of situations.

Curses window
Data structures, which can be thought of as two-dimensional arrays of characters that represent screen displays. These data structures are manipulated with Curses functions.

cursor position
The line and column position on the screen denoted by the terminal’s cursor.

empty wide-character string
A wide-character string whose first element is a null wide-character code.

erase character
A special input character that deletes the last character in the current line, if there is one.

kill character
A special input character that deletes all data in the current line, if there are any.

null chtype
A chtype with all bits set to zero.

null wide-character code
A wide-character code with all bits set to zero.

pad
A window that is not necessarily associated with a viewable part of a screen.

parent window
A window that has subwindows or derived windows associated with it.

rendition
The rendition of a character displayed on the screen is its attributes and a color pair.

SCREEN
An opaque Curses data type that is associated with the display screen.

subwindow
A window, created within another window, but positioned relative to that other window. Changes made to a subwindow do not affect its parent window. A derived window differs from a subwindow only in that it is positioned relative to the origin of its parent window. Changes to a parent window will affect both subwindows and derived windows.

touch
To set a flag in a window that indicates that the information in the window could differ from the that displayed on the terminal device.

wide-character code (C language)
An integer value corresponding to a single graphic symbol or control code.

wide-character string
A contiguous sequence of wide-character codes terminated by and including the first null wide-character code.

window
A two-dimensional array of characters representing all or part of the terminal screen. The term window in this document means one of the data structures maintained by the Curses implementation, unless specified otherwise. (This document does not define the interaction between the Curses implementation and other windowing system paradigms.)

window hierarchy
The aggregate of a parent window and all of its subwindows and derived windows.
Index

A
add_wch interface for enhanced curses 17
add_wchnstr interface for enhanced curses 17
addch interface for curses 13
addchstr interface for enhanced curses 15
addnstr interface for enhanced curses 16
alternate character sets 176
application usage 180
area clears 159
attr_get interface for enhanced curses 19
attroff interface for curses 18

B
basic capabilities 156
baudrate interface for curses 20
beep interface for curses 21
bkgd interface for enhanced curses 21, 22
bkgrnd interface for enhanced curses 23
border interface for enhanced curses 24
border_set curses for enhanced curses 25
box interface for curses 26
box_set interface for enhanced curses 27
buffer size 179

c

can_change_color interface for enhanced curses 27
capabilities that cause movement 172
cbreak interface for curses 29
chtag interface for enhanced curses 30
clear interface for curses 31
clearok interface for curses 32
crltobot interface for curses 33
crtoeol interface for curses 34
color manipulation 166
color_content interface for enhanced curses 34
color_pairs interface for enhanced curses 34
cols interface for enhanced curses 35
connexions for device aliases 180
copywin interface for curses 35
cur_term interface for enhanced curses 37
current window structure 8
curs_set interface for enhanced curses 36
curscr interface for curses 36
curses environment, windows 7
curses interfaces 13
curses library 1
curses.h header 125
cursor motions 158
default window structure 7
defined capabilities 144
del_curterm interface for enhanced curses 39
delay_output interface for curses 38
delays 164
delch interface for curses 38
deltekn interface for curses 41
delscreen interface for curses 41
delwin interface for curses 42
derwin interface for curses 42
device capabilities 156
dot-matrix graphics 177
doupdate interface for curses 43
dupwin interface for enhanced curses 44

echo interface for curses 44
echo_wchar interface for enhanced curses 46
echochar interface for enhanced curses 45
effect of changing printing resolution 178
endwin interface for curses 46
erase interface for curses 47
erasechar interface for curses 47

E
filter interface for enhanced curses 48
flash interface for curses 49
flushing interface for curses 50
formal grammar 142

G
get_wch interface for enhanced curses 57
get_wstr interface for enhanced curses 58
getbegyx interface for curses 50
getbkgd interface for enhanced curses 51
getbkgrnd interface for enhanced curses 51
getbkgd interface for enhanced curses 51
getchar interface for enhanced curses 52
getch interface for curses 52
getmaxyx interface for enhanced curses 54
gtn_wstr interface for enhanced curses 55

H
halfdelay interface for enhanced curses 59
has_colors interface for enhanced curses 60
has_ic interface for curses 60
headers 125
highlighting 161
hline interface for enhanced curses 61
hline_set interface for enhanced curses 62

I
idcok interface for enhanced curses 63
idlok interface for curses 63
immedok interface for enhanced curses 63
in_wch interface for enhanced curses 75
in_wchstr interface for enhanced curses 76
inch interface for curses 64
inchnstr interface for enhanced curses 65
init_color interface for enhanced curses 66
initializing curses 5
initscr interface for curses 66, 67
innstr interface for enhanced curses 67
innstr interface for enhanced curses 68
ins_nwstr interface for enhanced curses 72
ins_wch interface for enhanced curses 74
ins_wstr interface for enhanced curses 74
insch interface for curses 69
insdel1 interface for enhanced curses 70
insert/delete character 160
insert/delete line 159
insertln interface for curses 71
ins_nwstr interface for enhanced curses 71
insstr interface for enhanced curses 71
insstr interface for enhanced curses 73
intrflush-interface for curses 75
insstr interface for enhanced curses 75
insw interface for enhanced curses 77
is_linenotouched interface for curses 78
isendwin interface for enhanced curses 77

© Copyright IBM Corp. 1996, 2013

189
K

keyname interface for curses 79
keypad 163
keypad interface for curses 80
killchar interface for curses 80

L

leaveok interface for curses 81
line graphics 165
lines interface for enhanced curses 81
longname interface for curses 81

M

mainframe
education xv
manipulating window data 11
meta interface for enhanced curses 82
minimum guaranteed limits 142
move interface for curses 83
mv interface for curses 83
mvcur interface for enhanced curses 84
mvderwin interface for enhanced curses 85
mvprintw interface for curses 86
mvscanw interface for curses 86
mvwin interface for curses 87

N

naming conventions 2
napms interface for curses 88
newpad interface for curses 88
newterm interface for curses 89
nl interface for curses 90
no interface for curses 90
nodelay interface for curses 91
noqiflush interface for enhanced curses 91
Notices 183
notimeout interface for enhanced curses 92

O

overlay interface for curses 93

P

pads 9
pair_content interface for enhanced curses 94
parameterized strings 157
pechochar interface for enhanced curses 94
poutright interface for enhanced curses 95
print quality 179
printer capabilities 169
printer resolution 170
printing rate 179
printwin interface for curses 95
putp interface for enhanced curses 95
putwin interface for enhanced curses 96
qiflush interface for enhanced curses 97
refresh interface for curses 97
reset_prog_mode interface for curses 98
resetty interface for curses 98
resetterm interface for enhanced curses 99
ripoffline interface for enhanced curses 99
rounding values 170
sample entry 153
savetty interface for curses 100
scr_interface for curses 100
scr_dump interface for enhanced curses 101
scrl_interface for curses 102
scrolllock interface for curses 102
selecting a terminal 180
set_curterm interface for enhanced curses 103
set_term interface for curses 104
set_term_interface for curses 104
setscrreg interface for curses 104
setupterm interface for enhanced curses 104
similar terminals 169
slk_attroff interface for enhanced curses 105
source file syntax 141
special cases 169
specifying printer resolution 170
standend interface for curses 107
start_color interface for enhanced curses 108
status lines 164
stdscr_interface for enhanced cursors 108
structure of a curses program 3
subpad interface for enhanced curses 108
subwin interface for curses 108
subwindows 8
syncok interface for enhanced curses 109
tabs and initialization 164
term.h header for enhanced curses 139
termattrs interface for enhanced curses 110
termattr source format 141
terminfo source format 141
terminology 1

termname interface for enhanced curses 110
tgetent interface for enhanced curses 111
tigetflag interface for enhanced curses 112
timeout interface for enhanced curses 113
touchline interface for curses 113
tparm interface for enhanced curses 114
tputs interface for enhanced curses 114
trademarks 185
typeahead interface for enhanced curses 114
types of capabilities in the sample entry 154

U

unctrl header 140
unctrl interface for curses 115
underlining 161
ungech interface for enhanced curses 115
untouchwin interface for enhanced curses 116
use_env interface for enhanced curses 116

V

variations of terminal definitions 181
vidattr interface for enhanced curses 117
visible bells 161
vline interface for enhanced curses 118
vline_set interface for enhanced curses 118
vw_printw interface for enhanced curses 119
vw_scanw interface for enhanced curses 120
vwprintw interface for enhanced curses 120

W

w interface for curses 121
wunctrl interface for enhanced curses 123

Z

z/OS Basic Skills information center xv
Product Number: 5650-ZOS

Printed in USA