Technical Standard

X/Open Curses
Issue 4, Version 2

THE Open GROUP
[This page intentionally left blank]
X/Open CAE Specification

X/Open Curses, Issue 4, Version 2

X/Open Company Ltd.
© July 1996, X/Open Company Limited
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the copyright owners.

Portions of this document are derived from copyrighted material owned by Hewlett-Packard Company, International Business Machines Corporation, Novell Inc., The Open Software Foundation, and Sun Microsystems, Inc.

X/Open CAE Specification
X/Open Curses, Issue 4, Version 2
X/Open Document Number: C610

Published by X/Open Company Ltd., U.K.

Any comments relating to the material contained in this document may be submitted to X/Open at:

X/Open Company Limited
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:
XoSpecs@xopen.org
# Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>1</th>
<th>Introduction ..................................................................................................................</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td></td>
<td>This Document ................................................................................................................</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1</td>
<td></td>
<td>Relationship to Issue 3 ...............................................................................................</td>
<td>1</td>
</tr>
<tr>
<td>1.1.2</td>
<td></td>
<td>New Features ..................................................................................................................</td>
<td>1</td>
</tr>
<tr>
<td>1.1.3</td>
<td></td>
<td>To Be Withdrawn ...........................................................................................................</td>
<td>1</td>
</tr>
<tr>
<td>1.1.4</td>
<td></td>
<td>Withdrawn ....................................................................................................................</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td></td>
<td>Conformance ..................................................................................................................</td>
<td>1</td>
</tr>
<tr>
<td>1.2.1</td>
<td></td>
<td>Base Curses Conformance ............................................................................................</td>
<td>1</td>
</tr>
<tr>
<td>1.2.2</td>
<td></td>
<td>Enhanced Curses Conformance ......................................................................................</td>
<td>1</td>
</tr>
<tr>
<td>1.3</td>
<td></td>
<td>Terminology ..................................................................................................................</td>
<td>1</td>
</tr>
<tr>
<td>1.3.1</td>
<td></td>
<td>Shaded Text ..................................................................................................................</td>
<td>1</td>
</tr>
<tr>
<td>1.4</td>
<td></td>
<td>Format of Entries ........................................................................................................</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>2</th>
<th>Use and Implementation of Interfaces ..........................................................................</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td></td>
<td>C Language Definition ..................................................................................................</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td></td>
<td>The Compilation Environment .......................................................................................</td>
<td>8</td>
</tr>
<tr>
<td>2.2.1</td>
<td></td>
<td>The X/Open Name Space (ENHANCED CURSES) ................................................................</td>
<td>9</td>
</tr>
<tr>
<td>2.2.2</td>
<td></td>
<td>Interfaces Implemented as Macros (ENHANCED CURSES) .............................................</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td></td>
<td>Relationship to the XSH Specification .......................................................................</td>
<td>11</td>
</tr>
<tr>
<td>2.3.1</td>
<td></td>
<td>Error Numbers ..............................................................................................................</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td></td>
<td>Data Types ...................................................................................................................</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>3</th>
<th>Interface Overview .......................................................................................................</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td></td>
<td>Components .................................................................................................................</td>
<td>13</td>
</tr>
<tr>
<td>3.2</td>
<td></td>
<td>Screens, Windows and Terminals ................................................................................</td>
<td>14</td>
</tr>
<tr>
<td>3.3</td>
<td></td>
<td>Characters ...................................................................................................................</td>
<td>14</td>
</tr>
<tr>
<td>3.3.1</td>
<td></td>
<td>Character Storage Size ..............................................................................................</td>
<td>16</td>
</tr>
<tr>
<td>3.3.2</td>
<td></td>
<td>Multi-column Characters ............................................................................................</td>
<td>16</td>
</tr>
<tr>
<td>3.3.3</td>
<td></td>
<td>Attributes ....................................................................................................................</td>
<td>16</td>
</tr>
<tr>
<td>3.3.4</td>
<td></td>
<td>Rendition ....................................................................................................................</td>
<td>17</td>
</tr>
<tr>
<td>3.3.5</td>
<td></td>
<td>Non-spacing Characters .............................................................................................</td>
<td>17</td>
</tr>
<tr>
<td>3.3.6</td>
<td></td>
<td>Window Properties .....................................................................................................</td>
<td>17</td>
</tr>
<tr>
<td>3.4</td>
<td></td>
<td>Conceptual Operations ...............................................................................................</td>
<td>17</td>
</tr>
<tr>
<td>3.4.1</td>
<td></td>
<td>Screen Addressing ......................................................................................................</td>
<td>17</td>
</tr>
<tr>
<td>3.4.2</td>
<td></td>
<td>Basic Character Operations .......................................................................................</td>
<td>17</td>
</tr>
<tr>
<td>3.4.3</td>
<td></td>
<td>Special Characters .....................................................................................................</td>
<td>17</td>
</tr>
<tr>
<td>3.4.4</td>
<td></td>
<td>Rendition of Characters Placed into a Window ........................................................</td>
<td>17</td>
</tr>
<tr>
<td>3.5</td>
<td></td>
<td>Input Processing ........................................................................................................</td>
<td>17</td>
</tr>
<tr>
<td>3.5.1</td>
<td></td>
<td>Keypad Processing ....................................................................................................</td>
<td>17</td>
</tr>
<tr>
<td>3.5.2</td>
<td></td>
<td>Input Mode .................................................................................................................</td>
<td>17</td>
</tr>
<tr>
<td>3.5.3</td>
<td></td>
<td>Delay Mode ...............................................................................................................</td>
<td>17</td>
</tr>
<tr>
<td>3.5.4</td>
<td></td>
<td>Echo Processing .......................................................................................................</td>
<td>17</td>
</tr>
</tbody>
</table>

X/Open Curses, Issue 4, Version 2
Chapter 4 Curses Interfaces

3.6.1 Function Name Conventions ................................................................. 26
3.6.2 Function Families Provided................................................................. 27
3.7 Interfaces Implemented as Macros ......................................................... 29
3.8 Initialised Curses Environment .............................................................. 29
3.9 Synchronous and Networked Asynchronous Terminals ....................... 30

Curses Interfaces ..................................................................................... 31
addch() ........................................................................................................ 32
addchstr() .................................................................................................... 33
addnstr() ....................................................................................................... 34
addnwstr() .................................................................................................... 35
addwch() ....................................................................................................... 36
addwchstr() .................................................................................................. 37
attroff() ......................................................................................................... 38
attr_get() ...................................................................................................... 39
baudrate() ...................................................................................................... 41
beep() ............................................................................................................. 42
bkgd() ............................................................................................................. 43
bkgrnd() ......................................................................................................... 44
border() ......................................................................................................... 45
border_set() .................................................................................................. 46
box() ............................................................................................................... 47
box_set() ........................................................................................................ 48
can_change_color() ..................................................................................... 49
cbreak() .......................................................................................................... 52
delch() ............................................................................................................. 53
delcurterm() ................................................................................................. 54
delchstr() ....................................................................................................... 55
delnstr() ......................................................................................................... 57
delwchstr() ..................................................................................................... 58
colour_content() .......................................................................................... 59
COLOR_PAIRS ................................................................................................. 60
COLS .................................................................................................................. 61
copywin() ....................................................................................................... 62
curscr ............................................................................................................... 63
curs_set() ....................................................................................................... 64
cur_term .......................................................................................................... 65
def_prog_mode() ........................................................................................... 66
delay_output() ............................................................................................... 67
delwin() ........................................................................................................... 68
delcurterm() ................................................................................................. 69
deleteln() ....................................................................................................... 71
delscreen() ..................................................................................................... 72
delwch() ......................................................................................................... 73
derwin() .......................................................................................................... 74
doupdate() ....................................................................................................... 75
dupwin() .......................................................................................................... 76
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>echo()</td>
<td>77</td>
</tr>
<tr>
<td>echochar()</td>
<td>78</td>
</tr>
<tr>
<td>echo_wchar()</td>
<td>79</td>
</tr>
<tr>
<td>endwin()</td>
<td>80</td>
</tr>
<tr>
<td>erase()</td>
<td>81</td>
</tr>
<tr>
<td>erasechar()</td>
<td>82</td>
</tr>
<tr>
<td>filter()</td>
<td>83</td>
</tr>
<tr>
<td>flash()</td>
<td>84</td>
</tr>
<tr>
<td>flushinp()</td>
<td>85</td>
</tr>
<tr>
<td>getbegyx()</td>
<td>86</td>
</tr>
<tr>
<td>getbkgd()</td>
<td>88</td>
</tr>
<tr>
<td>getbkgnd()</td>
<td>89</td>
</tr>
<tr>
<td>getcchar()</td>
<td>90</td>
</tr>
<tr>
<td>getch()</td>
<td>91</td>
</tr>
<tr>
<td>getmaxyx()</td>
<td>93</td>
</tr>
<tr>
<td>getnstr()</td>
<td>94</td>
</tr>
<tr>
<td>getn_wstr()</td>
<td>96</td>
</tr>
<tr>
<td>getparyx()</td>
<td>98</td>
</tr>
<tr>
<td>getstr()</td>
<td>99</td>
</tr>
<tr>
<td>get_wch()</td>
<td>100</td>
</tr>
<tr>
<td>getwin()</td>
<td>101</td>
</tr>
<tr>
<td>get_wstr()</td>
<td>102</td>
</tr>
<tr>
<td>getyx()</td>
<td>103</td>
</tr>
<tr>
<td>halfdelay()</td>
<td>104</td>
</tr>
<tr>
<td>has_colors()</td>
<td>105</td>
</tr>
<tr>
<td>has_ic()</td>
<td>106</td>
</tr>
<tr>
<td>hline()</td>
<td>107</td>
</tr>
<tr>
<td>hline_set()</td>
<td>108</td>
</tr>
<tr>
<td>idcok()</td>
<td>109</td>
</tr>
<tr>
<td>idlok()</td>
<td>110</td>
</tr>
<tr>
<td>immedok()</td>
<td>111</td>
</tr>
<tr>
<td>inch()</td>
<td>112</td>
</tr>
<tr>
<td>inchnstr()</td>
<td>113</td>
</tr>
<tr>
<td>init_color()</td>
<td>114</td>
</tr>
<tr>
<td>initscr()</td>
<td>115</td>
</tr>
<tr>
<td>innstr()</td>
<td>117</td>
</tr>
<tr>
<td>innwstr()</td>
<td>118</td>
</tr>
<tr>
<td>insch()</td>
<td>119</td>
</tr>
<tr>
<td>insdelin()</td>
<td>120</td>
</tr>
<tr>
<td>insertln()</td>
<td>121</td>
</tr>
<tr>
<td>insnstr()</td>
<td>122</td>
</tr>
<tr>
<td>ins_nwstr()</td>
<td>123</td>
</tr>
<tr>
<td>insstr()</td>
<td>124</td>
</tr>
<tr>
<td>instr()</td>
<td>125</td>
</tr>
<tr>
<td>ins_wch()</td>
<td>126</td>
</tr>
<tr>
<td>ins_wstr()</td>
<td>127</td>
</tr>
<tr>
<td>intrflush()</td>
<td>128</td>
</tr>
<tr>
<td>in_wch()</td>
<td>129</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------</td>
</tr>
<tr>
<td>in_vchnstr</td>
<td>130</td>
</tr>
<tr>
<td>inwstr</td>
<td>131</td>
</tr>
<tr>
<td>isendwin</td>
<td>132</td>
</tr>
<tr>
<td>is_linetouched</td>
<td>133</td>
</tr>
<tr>
<td>keyname</td>
<td>134</td>
</tr>
<tr>
<td>keypad</td>
<td>135</td>
</tr>
<tr>
<td>killchar</td>
<td>136</td>
</tr>
<tr>
<td>leaveok</td>
<td>137</td>
</tr>
<tr>
<td>LINES</td>
<td>138</td>
</tr>
<tr>
<td>longname</td>
<td>139</td>
</tr>
<tr>
<td>meta</td>
<td>140</td>
</tr>
<tr>
<td>move</td>
<td>141</td>
</tr>
<tr>
<td>mv</td>
<td>142</td>
</tr>
<tr>
<td>mocur</td>
<td>144</td>
</tr>
<tr>
<td>mderwin</td>
<td>145</td>
</tr>
<tr>
<td>mpprintw</td>
<td>146</td>
</tr>
<tr>
<td>mslace</td>
<td>147</td>
</tr>
<tr>
<td>mvw</td>
<td>148</td>
</tr>
<tr>
<td>napms</td>
<td>149</td>
</tr>
<tr>
<td>newpad</td>
<td>150</td>
</tr>
<tr>
<td>newterm</td>
<td>152</td>
</tr>
<tr>
<td>newwin</td>
<td>153</td>
</tr>
<tr>
<td>nl</td>
<td>154</td>
</tr>
<tr>
<td>no</td>
<td>155</td>
</tr>
<tr>
<td>nodelay</td>
<td>156</td>
</tr>
<tr>
<td>noqiflush</td>
<td>157</td>
</tr>
<tr>
<td>notimeout</td>
<td>158</td>
</tr>
<tr>
<td>overlay</td>
<td>159</td>
</tr>
<tr>
<td>pair_content</td>
<td>160</td>
</tr>
<tr>
<td>pechochar</td>
<td>161</td>
</tr>
<tr>
<td>pnoutrefresh</td>
<td>162</td>
</tr>
<tr>
<td>printw</td>
<td>163</td>
</tr>
<tr>
<td>putp</td>
<td>164</td>
</tr>
<tr>
<td>putwin</td>
<td>165</td>
</tr>
<tr>
<td>qiflush</td>
<td>166</td>
</tr>
<tr>
<td>raw</td>
<td>167</td>
</tr>
<tr>
<td>redrawwin</td>
<td>168</td>
</tr>
<tr>
<td>refresh</td>
<td>169</td>
</tr>
<tr>
<td>reset_prog_mode</td>
<td>170</td>
</tr>
<tr>
<td>resetty</td>
<td>171</td>
</tr>
<tr>
<td>restartterm</td>
<td>172</td>
</tr>
<tr>
<td>ripoffline</td>
<td>173</td>
</tr>
<tr>
<td>savetty</td>
<td>174</td>
</tr>
<tr>
<td>scanw</td>
<td>175</td>
</tr>
<tr>
<td>scr_dump</td>
<td>176</td>
</tr>
<tr>
<td>scrl</td>
<td>177</td>
</tr>
<tr>
<td>scrollok</td>
<td>178</td>
</tr>
<tr>
<td>setcchar</td>
<td>179</td>
</tr>
</tbody>
</table>
set_curterm() ........................................................................................................ 180
setscrreg() ........................................................................................................ 181
set_term() .......................................................................................................... 182
setupterm() ....................................................................................................... 183
silk_attroff() ................................................................................................. 184
standend() ....................................................................................................... 186
start_color() ...................................................................................................... 187
stdscr .................................................................................................................. 188
subpad() ............................................................................................................. 189
subwin() ............................................................................................................. 190
syncok() ............................................................................................................. 191
termattrs() ........................................................................................................ 192
termname() ........................................................................................................ 193
tgetent() ............................................................................................................ 194
tigetflag() ......................................................................................................... 196
timeout() ............................................................................................................ 198
touchline() ........................................................................................................ 199
tparm() ............................................................................................................... 200
tputs() ................................................................................................................. 201
typeahead() ...................................................................................................... 202
unctrl() .............................................................................................................. 203
ungetch() .......................................................................................................... 204
untouchwin() .................................................................................................... 205
use_env() .......................................................................................................... 206
vidattr() ............................................................................................................. 207
vline() ............................................................................................................... 209
vline_set() ....................................................................................................... 210
vwprintw() ....................................................................................................... 211
vw_printw() .................................................................................................... 212
vwscanw() ....................................................................................................... 213
vw_scanw() ....................................................................................................... 214
w ......................................................................................................................... 215
wunctrl() .......................................................................................................... 218

Chapter 5 Headers ............................................................................................. 219
   <curses.h> ...................................................................................................... 220
   <term.h> ....................................................................................................... 235
   <unctrl.h> .................................................................................................... 236

Chapter 6 Terminfo Source Format (ENHANCED CURSES) .................. 237
6.1 Source File Syntax ....................................................................................... 238
6.1.1 Minimum Guaranteed Limits ............................................................... 239
6.1.2 Formal Grammar ................................................................................... 239
6.1.3 Defined Capabilities ............................................................................... 241
6.1.4 Sample Entry .......................................................................................... 251
6.1.5 Types of Capabilities in the Sample Entry ........................................ 251
X/Open

X/Open is an independent, worldwide, open systems organisation supported by most of the world’s largest information systems suppliers, user organisations and software companies. Its mission is to bring to users greater value from computing, through the practical implementation of open systems.

X/Open's strategy for achieving this goal is to combine existing and emerging standards into a comprehensive, integrated, high-value and usable open system environment, called the Common Applications Environment (CAE). This environment covers the standards, above the hardware level, that are needed to support open systems. It provides for portability and interoperability of applications, and so protects investment in existing software while enabling additions and enhancements. It also allows users to move between systems with a minimum of retraining.

X/Open defines this CAE in a set of specifications which include an evolving portfolio of application programming interfaces (APIs) which significantly enhance portability of application programs at the source code level, along with definitions of and references to protocols and protocol profiles which significantly enhance the interoperability of applications and systems.

The X/Open CAE is implemented in real products and recognised by a distinctive trade mark — the X/Open brand — that is licensed by X/Open and may be used on products which have demonstrated their conformance.

X/Open Technical Publications

X/Open publishes a wide range of technical literature, the main part of which is focussed on specification development, but which also includes Guides, Snapshots, Technical Studies, Branding/Testing documents, industry surveys, and business titles.

There are two types of X/Open specification:

- **CAE Specifications**

  CAE (Common Applications Environment) specifications are the stable specifications that form the basis for X/Open-branded products. These specifications are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement an X/Open CAE specification can enjoy the benefits of a single, widely supported standard. In addition, they can demonstrate compliance with the majority of X/Open CAE specifications once these specifications are referenced in an X/Open component or profile definition and included in the X/Open branding programme.

  CAE specifications are published as soon as they are developed, not published to coincide with the launch of a particular X/Open brand. By making its specifications available in this way, X/Open makes it possible for confromant products to be developed as soon as is practicable, so enhancing the value of the X/Open brand as a procurement aid to users.
• **Preliminary Specifications**

These specifications, which often address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations, are released in a controlled manner for the purpose of validation through implementation of products. A Preliminary specification is not a draft specification. In fact, it is as stable as X/Open can make it, and on publication has gone through the same rigorous X/Open development and review procedures as a CAE specification.

Preliminary specifications are analogous to the *trial-use* standards issued by formal standards organisations, and product development teams are encouraged to develop products on the basis of them. However, because of the nature of the technology that a Preliminary specification is addressing, it may be untried in multiple independent implementations, and may therefore change before being published as a CAE specification. There is always the intent to progress to a corresponding CAE specification, but the ability to do so depends on consensus among X/Open members. In all cases, any resulting CAE specification is made as upwards-compatible as possible. However, complete upwards-compatibility from the Preliminary to the CAE specification cannot be guaranteed.

In addition, X/Open publishes:

• **Guides**

These provide information that X/Open believes is useful in the evaluation, procurement, development or management of open systems, particularly those that are X/Open-compliant. X/Open Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming X/Open conformance.

• **Technical Studies**

X/Open Technical Studies present results of analyses performed by X/Open on subjects of interest in areas relevant to X/Open’s Technical Programme. They are intended to communicate the findings to the outside world and, where appropriate, stimulate discussion and actions by other bodies and the industry in general.

• **Snapshots**

These provide a mechanism for X/Open to disseminate information on its current direction and thinking, in advance of possible development of a Specification, Guide or Technical Study. The intention is to stimulate industry debate and prototyping, and solicit feedback. A Snapshot represents the interim results of an X/Open technical activity. Although at the time of its publication, there may be an intention to progress the activity towards publication of a Specification, Guide or Technical Study, X/Open is a consensus organisation, and makes no commitment regarding future development and further publication. Similarly, a Snapshot does not represent any commitment by X/Open members to develop any specific products.

**Versions and Issues of Specifications**

As with all *live* documents, CAE Specifications require revision, in this case as the subject technology develops and to align with emerging associated international standards. X/Open makes a distinction between revised specifications which are fully backward compatible and those which are not:

• a new *Version* indicates that this publication includes all the same (unchanged) definitive information from the previous publication of that title, but also includes extensions or additional information. As such, it *replaces* the previous publication.
• a new Issue does include changes to the definitive information contained in the previous publication of that title (and may also include extensions or additional information). As such, X/Open maintains both the previous and new issue as current publications.

Corrigenda

Most X/Open publications deal with technology at the leading edge of open systems development. Feedback from implementation experience gained from using these publications occasionally uncovers errors or inconsistencies. Significant errors or recommended solutions to reported problems are communicated by means of Corrigenda.

The reader of this document is advised to check periodically if any Corrigenda apply to this publication. This may be done in any one of the following ways:

• anonymous ftp to ftp.xopen.org
• ftpmail (see below)
• reference to the Corrigenda list in the latest X/Open Publications Price List.

To request Corrigenda information using ftpmail, send a message to ftpmail@xopen.org with the following four lines in the body of the message:

open
cd pub/Corrigenda
get index
quit

This will return the index of publications for which Corrigenda exist. Use the same email address to request a copy of the full corrigendum information following the email instructions.

This Document

This specification is a CAE Specification (see above) that defines the X/Open Curses interface offered to application programs by X/Open Curses conformant systems. Readers are expected to be experienced C language programmers and to be familiar with the XBD specification. This specification is structured as follows:

• Chapter 1 introduces Curses, gives an overview of enhancements that have been made to this version and lists specific interfaces marked TO BE WITHDRAWN. This chapter also defines the requirements for conformance to this document and shows the generic format followed by interface definitions in Chapter 4.
• Chapter 2 describes the relationship between Curses and the C language, the compilation environment, and the X/Open System Interface operating system requirements. It also defines the effect of the interface on the name space for identifiers and introduces the major data types that the interfaces use.
• Chapter 3 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 4. 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 4 defines the Curses functional interfaces.
• Chapter 5 defines the contents of headers which declare constants, macros and data structures that are needed by programs using the services provided by Chapter 4.

• Chapter 6 on page 237 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.

• Appendix A on page 255 discusses the use of these capabilities by the writer of a terminfo entry to describe the characteristics of the terminal in use.

The chapters are followed by a glossary, which contains normative definitions of terms used in the document. Comprehensive references are available in the index.

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

• Shading is used to identify X/Open Enhanced Curses material, relating to interfaces included to provide enhanced capabilities for applications originally written to be compiled on systems based on the UNIX operating system. Therefore, the features described may not be present on systems that conform to XPG4 or to earlier XPG releases. The relevant reference manual pages may provide additional or more specific portability warnings about use of the material.

If an entire SYNOPSIS section is shaded and marked with one EC, all the functionality described in that entry is an extension.

The material on pages labelled ENHANCED CURSES and the material flagged with the EC margin legend is available only in cases where the _XOPEN_CURSES version test macro is defined.

Notes:

1. 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>.
2. The values of errors are defined in <errno.h>.
Trade Marks

AT&T® is a registered trade mark of AT&T in the U.S.A. and other countries.

HP® is a registered trade mark of Hewlett-Packard.

UNIX® is a registered trade mark in the United States and other countries, licensed exclusively through X/Open Company Limited.

X/Open® is a registered trade mark, and the “X” device is a trade mark, of X/Open Company Limited.

The names of terminals and of terminal manufacturers cited as examples in Chapter 6 and Appendix A may be trade marks, which are the property of their respective owners.
Acknowledgements

X/Open gratefully acknowledges:

- Novell, Inc. for permission to reproduce portions of its copyrighted System V Interface Definition (SVID) and material from the UNIX System V Release 4.2 documentation.
- Hewlett-Packard Company, International Business Machines Corporation, Novell Inc., The Open Software Foundation, and Sun Microsystems, Inc., for their work in developing the X/Open UNIX Extension and sponsoring it through the X/Open Direct Review (Fast-track) process.
Referenced Documents

The following documents are referenced in this specification:

ANSI C

ISO 8859-1

ISO/IEC 646

ISO C
ISO/IEC 9899: 1990: Programming Languages — C, including:
Amendment 1: 1994, Multibyte Support Extensions (MSE) for ISO C.
Amendment 1: 1995, C Integrity.

SVID Issue 2
System V Interface Definition (Spring 1986 - Issue 2).

SVID 3rd Edition

System V Release 2.0

System V Release 4.2

The following X/Open documents are referenced in this specification.

Curses Interface, Issue 3

Curses Interface, Issue 4

Curses Interface, Issue 4, Version 2

Headers Interface
Referenced Documents


XBD, Issue 4, Version 2

XCU, Issue 4, Version 2

XSH, Issue 3

XSH, Issue 4

XSH, Issue 4, Version 2
Introduction

The Curses interface provides a terminal-independent method of updating character screens. The functions in this document are oriented towards locally-connected asynchronous terminals that recognise the code set of the current locale. For such terminals, applications conforming to this interface are portable. The Curses interface may also be used with synchronous and networked asynchronous terminals, provided the restrictions described in Section 3.9 on page 30 are considered.

1.1 This Document


1.1.1 Relationship to Issue 3

The unshaded material in this specification preserves syntactic compatibility with the Curses specification, Issue 3, although existing interfaces from the Curses specification, Issue 3 have been clarified as a result of industry feedback.

Relationship to Issue 4, Version 1

Version 2 contains corrections and clarifications which have been suggested by industry feedback. In particular, many of the function prototypes have been corrected, and colour handling has been further clarified. The CHANGE HISTORY section of the manpages gives specific detail on when changes were made.

1.1.2 New Features

These are the features first introduced for issue 4.

Internationalisation

This version of the Curses specification has been enhanced to support a wide range of internationalised capabilities. Traditional single-byte character operations are preserved, and multi-byte and wide-character interfaces are included to allow use of the Curses features with a wide range of character code sets. The actual code sets supported are implementation-defined.

Enhanced Character Sets

Emerging character-set standards specify characters with a constant width greater than an octet (such as ISO/IEC 10646-1:1993), or multi-byte code sets (such as the ISO 2022:1986 EUC encoding used to encode the Japanese and Chinese language characters).

The previous version of the Curses specification was capable of supporting ISO 8859-1. Many traditional implementations only supported ISO 646:1991 and preceding code set specifications, in which the length of a character was an octet.

The primary standardisation issue with the increasing size of a character is that neither the ANS X3.159-1989 nor ISO/IEC 9899:1990 C language definition requires the existence of an integral data type greater than 32-bits. Although such data types are commonly defined, X/Open cannot require support for them at this time. The opaque data type cchar_t and associated routines address this issue.
Writing Direction

The references to writing direction have been generalised to permit both right-to-left and left-to-right writing. This specification does not specify whether the implementation supports more than one direction of writing. The behaviour of the interfaces in this volume is unspecified if the writing direction is vertical, or if the writing direction is horizontal with row height greater than one.

Wide and Non-spacing Characters

New interfaces are introduced for use with wide characters and wide character strings. The traditional single-byte character string interfaces have been made more general for use with multi-byte character strings. The traditional chtype interfaces note that they are usable only in restricted environments and do not support extensible attributes. The behaviour of the chtype interfaces in this volume is unspecified if the char data type is greater than 8 bits, or if any single byte character takes more than one display column, or if the application or implementation stores a multi-byte or wide-character value into a chtype object.

A new, extensible attribute model has been provided for wide-character interfaces. The display model has been generalised to support both multi-column characters and non-spacing characters. The concept of a complex character is introduced.

Other Enhancements

New interfaces and capabilities are introduced to support colour terminals, printers, modems and mice.

1.1.3 To Be Withdrawn

Some of the interfaces and headers in this issue are marked TO BE WITHDRAWN. Various factors may have contributed to the decision to withdraw an interface. In all cases, the reasons for withdrawal of an interface are documented on the relevant pages.

If a migration path exists, advice is given to application developers regarding alternative means of obtaining similar functionality. This information may be found in the APPLICATION USAGE sections on the relevant pages in Chapter 4.

Interfaces marked TO BE WITHDRAWN shall still exist on conformant implementations. However, they will be marked WITHDRAWN in a future issue of this document. Interfaces marked WITHDRAWN may still exist on conformant implementations.

Application writers should not use functionality marked TO BE WITHDRAWN.
The following interfaces are marked TO BE WITHDRAWN in this document:

<table>
<thead>
<tr>
<th>Headers and Interfaces To Be Withdrawn</th>
</tr>
</thead>
<tbody>
<tr>
<td>tgetent() tgetnum() tgoto() vwscanw()</td>
</tr>
<tr>
<td>tgetflag() tgetstr() vwprintw()</td>
</tr>
</tbody>
</table>

1.1.4 Withdrawn

No interfaces or headers in this issue are marked WITHDRAWN.

1.2 Conformance

An implementation conforming to this document shall meet the requirements specified by Base Curses conformance (see Section 1.2.1) or by Enhanced Curses conformance (see Section 1.2.2).

1.2.1 Base Curses Conformance

An implementation that claims Base Curses conformance shall meet the following criteria:

- The system shall support all the interfaces and headers defined within this specification except that it need not support those occurring on pages labelled ENHANCED CURSES and in shaded areas of this specification marked with the EC margin legend.
- The chtype data type shall support at least octet-based code sets, such as ISO 8859-1.
- The system may provide additional or enhanced interfaces, headers and facilities not required by this specification, provided that such additions or enhancements do not affect the behaviour of an application that requires only the facilities described in this document.

1.2.2 Enhanced Curses Conformance

An implementation that claims Enhanced Curses conformance shall meet the following criteria:

- The system shall support Base Curses conformance as defined above.
- The system shall support the requirements in this specification occurring on pages labelled ENHANCED CURSES and in shaded areas of this specification marked with the EC margin legend.
- The system may provide additional or enhanced interfaces, headers and facilities not required by this specification, provided that such additions or enhancements do not affect the behaviour of an application that requires only the facilities described in this document.
1.3 Terminology

The following terms are used in this specification:

**can**
This describes a permissible optional feature or behaviour available to the user or application; all systems support such features or behaviour as mandatory requirements.

**implementation-dependent**
The value or behaviour is not consistent across all implementations. The provider of an implementation normally documents the requirements for correct program construction and correct data in the use of that value or behaviour. When the value or behaviour in the implementation is designed to be variable or customisable on each instantiation of the system, the provider of the implementation normally documents the nature and permissible ranges of this variation. Applications that are intended to be portable must not rely on implementation-dependent values or behaviour.

**may**
With respect to implementations, the feature or behaviour is optional. Applications should not rely on the existence of the feature. To avoid ambiguity, the reverse sense of may is expressed as need not, instead of may not.

**must**
This describes a requirement on the application or user.

**obsolescent**
Certain features are obsolescent, which means that they may be considered for withdrawal in future revisions of this document. They are retained in this version because of their widespread use. Their use in new applications is discouraged.

**should**
With respect to implementations, the feature is recommended, but it is not mandatory. Applications should not rely on the existence of the feature.

With respect to users or applications, the word means recommended programming practice that is necessary for maximum portability.

**undefined**
A value or behaviour is undefined if this document imposes no portability requirements on applications for erroneous program constructs or erroneous data. Implementations may specify the result of using that value or causing that behaviour, but such specifications are not guaranteed to be consistent across all implementations. An application using such behaviour is not fully portable to all systems.

**unspecified**
A value or behaviour is unspecified if this document imposes no portability requirements on applications for correct program construct or correct data. Implementations may specify the result of using that value or causing that behaviour, but such specifications are not guaranteed to be consistent across all implementations. An application requiring a specific behaviour, rather than tolerating any behaviour when using that functionality, is not fully portable to all systems.

**will**
This means that the behaviour described is a requirement on the implementation and applications can rely on its existence.
1.3.1 Shaded Text

Shaded text in this document is qualified by a code in the left margin. The code and its meaning is as follows:

EC  X/Open Enhanced Curses

The material relates to interfaces included to provide enhanced capabilities for applications originally written to be compiled on systems based on the UNIX operating system. Therefore, the features described may not be present on systems that conform to XPG4 or to earlier XPG releases. The relevant reference manual pages may provide additional or more specific portability warnings about use of the material.

If an entire SYNOPSIS section is shaded and marked with one EC, all the functionality described in that entry is an extension.

The material on pages labelled ENHANCED CURSES and the material flagged with the EC margin legend is available only in cases where the _XOPEN_CURSES version test macro is defined.
1.4 Format of Entries

The entries in Chapter 4 and Chapter 5 are based on a common format.

NAME
This section gives the name or names of the entry and briefly states its purpose.

SYNOPSIS
This section summarises the use of the entry being described. If it is necessary to include a header to use this interface, the names of such headers are shown, for example:

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

DESCRIPTION
This section describes the functionality of the interface or header.

RETURN VALUE
This section indicates the possible return values, if any.

If the implementation can detect errors, “successful completion” means that no error has been detected during execution of the function. If the implementation does detect an error, the error will be indicated.

For functions where no errors are defined, “successful completion” means that if the implementation checks for errors, no error has been detected. If the implementation can detect errors, and an error is detected, the indicated return value will be returned and `errno` may be set.

ERRORS
This section gives the symbolic names of the values returned in `errno` if an error occurs. “No errors are defined” means that values and usage of `errno`, if any, depend on the implementation.

EXAMPLES
This section gives examples of usage, where appropriate.

APPLICATION USAGE
This section gives warnings and advice to application writers about the entry.

FUTURE DIRECTIONS
This section provides comments which should be used as a guide to current thinking; there is not necessarily a commitment to adopt these future directions.

SEE ALSO
This section gives references to related information.

CHANGE HISTORY
This section shows the derivation of the entry and any significant changes that have been made to it.

The only sections relating to conformance are the SYNOPSIS, DESCRIPTION, RETURN VALUE and ERRORS sections.
Each of the following statements applies unless explicitly stated otherwise in the detailed descriptions that follow. If an argument to a function has an invalid value (such as a value outside the domain of the function, or a pointer outside the address space of the program, or a null pointer), the behaviour is undefined. Any function declared in a header may also be implemented as a macro defined in the header, so a library function should not be declared explicitly if its header is included. Any macro definition of a function can be suppressed locally by enclosing the name of the function in parentheses, because the name is then not followed by the left parenthesis that indicates expansion of a macro function name. For the same syntactic reason, it is permitted to take the address of a library function even if it is also defined as a macro. The use of the C-language `#undef` construct to remove any such macro definition will also ensure that an actual function is referred to. Any invocation of a library function that is implemented as a macro will expand to code that evaluates each of its arguments exactly once, fully protected by parentheses where necessary, so it is generally safe to use arbitrary expressions as arguments. Likewise, those function-like macros described in the following sections may be invoked in an expression anywhere a function with a compatible return type could be called.

Provided that a library function can be declared without reference to any type defined in a header, it is also permissible to declare the function, either explicitly or implicitly, and use it without including its associated header. If a function that accepts a variable number of arguments is not declared (explicitly or by including its associated header), the behaviour is undefined.

As a result of changes introduced in this version of the Curses specification, application writers are only required to include the minimum number of headers. Implementations of XSI-conformant systems will make all necessary symbols visible as described in the Headers section of this document.

### 2.1 C Language Definition

The C language that is the basis for the synopses and code examples in this document is ISO C, as specified in the referenced ISO C standard. Common Usage C, which refers to the C language before standardisation, was the basis for previous editions of this specification.
2.2 **The Compilation Environment**

Applications should ensure that the feature test macro \_XOPEN\_SOURCE is defined before inclusion of any header. This is needed to enable the functionality described in this document, and possibly to enable functionality defined elsewhere in the Common Applications Environment.

The \_XOPEN\_SOURCE macro may be defined automatically by the compilation process, but to ensure maximum portability, applications should make sure that \_XOPEN\_SOURCE is defined by using either compiler options or \#define directives in the source files, before any \#include directives. Identifiers in this document may only be undefined using the \#undef directive as described in Chapter 2 on page 7 or Section 2.2.1 on page 9. These \#undef directives must follow all \#include directives of any XSI headers.

Most strictly conforming POSIX and ISO C applications will compile on systems compliant to this specification. However, an application which uses any of the items marked as an extension to POSIX and ISO C, for any purpose other than that shown here, may not compile. In such cases, it may be necessary to alter those applications to use alternative identifiers.

Since this document is aligned with the ISO C standard, and since all functionality enabled by the \_POSIX\_C\_SOURCE set equal to 2 should be enabled by \_XOPEN\_SOURCE, there should be no need to define either \_POSIX\_SOURCE or \_POSIX\_C\_SOURCE if \_XOPEN\_SOURCE is defined. Therefore if \_XOPEN\_SOURCE is defined and \_POSIX\_SOURCE is defined, or \_POSIX\_C\_SOURCE is set equal to 1 or 2, the behaviour is the same as if only \_XOPEN\_SOURCE is defined. However should \_POSIX\_C\_SOURCE be set to a value greater than 2, the behaviour is undefined.

The `c89` and `cc` utilities recognise the additional \-l operand for standard libraries:

\-l curses This operand makes visible all library functions referenced in this specification, (except for those labelled ENHANCED CURSES and except for portions marked with the EC margin legend).

If the implementation defines \_XOPEN\_CURSES and if the application defines the \_XOPEN\_SOURCE\_EXTENDED feature test macro, then \-l curses also makes visible all library functions referenced in this specification and labelled ENHANCED CURSES and portions marked with the EC margin legend.

It is unspecified whether the library libcurses.a exists as a regular file.

An application that uses any API specified as ENHANCED CURSES or relies on any portion of this specification marked with the EC margin legend must define \_XOPEN\_SOURCE\_EXTENDED = 1 in each source file or as part of its compilation environment. When \_XOPEN\_SOURCE\_EXTENDED = 1 is defined in a source file, it must appear before any header is included.

If the implementation supports the utilities marked DEVELOPMENT in the XCU specification, the lint utility recognises the additional \-l curses operand for standard libraries:

\-l curses Names the library llib-lcurses.ln, which will contain functions specified in this document.

It is unspecified whether the library llib-lcurses.ln exists as a regular file.
2.2.1 The X/Open Name Space (ENHANCED CURSES)

The requirements in this section are in effect only for implementations that claim Enhanced Curses compliance.

All identifiers in this document are defined in at least one of the headers, as shown in Chapter 5. When \_XOPEN\_SOURCE is defined, each header defines or declares some identifiers, potentially conflicting with identifiers used by the application. The set of identifiers visible to the application consists of precisely those identifiers from the header pages of the included headers, as well as additional identifiers reserved for the implementation. In addition, some headers may make visible identifiers from other headers as indicated on the relevant header pages.

The identifiers reserved for use by the implementation are described below.

1. Each identifier with external linkage described in the header section is reserved for use as an identifier with external linkage if the header is included.

2. Each macro name described in the header section is reserved for any use if the header is included.

3. Each identifier with file scope described in the header section is reserved for use as an identifier with file scope in the same name space if the header is included.

4. All identifiers consisting of exactly 2 upper-case letters.
If any header is included, identifiers with the _t suffix are reserved for any use by the implementation.

If any header in the following table is included, macros with the prefixes shown may be defined. After the last inclusion of a given header, an application may use identifiers with the corresponding prefixes for its own purpose, provided their use is preceded by an \texttt{#undef} of the corresponding macro.

<table>
<thead>
<tr>
<th>Header</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;curses.h&gt;</td>
<td>A_, ACS_, ALL_, BUTTON, COLOR_, KEY_, MOUSE, REPORT_, WA_, WACS_</td>
</tr>
<tr>
<td>&lt;term.h&gt;</td>
<td>ext_</td>
</tr>
</tbody>
</table>

The following identifiers are reserved regardless of the inclusion of headers:

1. All identifiers that begin with an underscore and either an upper-case letter or another underscore are always reserved for any use by the implementation.
2. All identifiers that begin with an underscore are always reserved for use as identifiers with file scope in both the ordinary identifier and tag name spaces.
3. All identifiers listed as reserved in the XSH specification are reserved for use as identifiers with external linkage.

All the identifiers defined in this document that have external linkage are always reserved for use as identifiers with external linkage.

No other identifiers are reserved.

Applications must not declare or define identifiers with the same name as an identifier reserved in the same context. Since macro names are replaced whenever found, independent of scope and name space, macro names matching any of the reserved identifier names must not be defined if any associated header is included.

Headers may be included in any order, and each may be included more than once in a given scope, with no difference in effect from that of being included only once.

If used, a header must be included outside of any external declaration or definition, and it must be first included before the first reference to any type or macro it defines, or to any function or object it declares. However, if an identifier is declared or defined in more than one header, the second and subsequent associated headers may be included after the initial reference to the identifier. Prior to the inclusion of a header, the program must not define any macros with names lexically identical to symbols defined by that header.
2.2.2 Interfaces Implemented as Macros (ENHANCED CURSES)

The requirements in this section are in effect only for implementations that claim Enhanced Curses compliance.

The following interfaces with arguments must be implemented as macros. The relevance to the application programmer is that the ‘&’ character cannot be used before the arguments.

<table>
<thead>
<tr>
<th>Macros</th>
<th>Chapter 4 Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR_PAIR(), PAIR_NUMBER()</td>
<td>can_change_color()</td>
</tr>
<tr>
<td>getbegyx(), getmaxyx(), getparyx(), getyx()</td>
<td>getbegyx()</td>
</tr>
</tbody>
</table>

The descriptions of headers in Chapter 5 list other macros, like COLOR_BLACK, that do not take arguments.

2.3 Relationship to the XSH Specification

2.3.1 Error Numbers

Most functions provide an error number in _errno, which is either a variable or macro defined in <errno.h>; the macro expands to a modifiable lvalue of type int.

A list of valid values for _errno and advice to application writers on the use of _errno appears in the XSH specification.
2.4 Data Types

All of the data types used by Curses functions are defined by the implementation. The following list describes these types:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr_t</td>
<td>An integral type that can contain at least an unsigned short. The type attr_t is used to hold an OR-ed set of attributes defined in &lt;curses.h&gt; that begin with the prefix WA_.</td>
</tr>
<tr>
<td>bool</td>
<td>Boolean data type</td>
</tr>
<tr>
<td>ctype</td>
<td>An integral type that can contain at least an unsigned char and attributes. Values of type ctype are formed by OR-ing together an unsigned char value and zero or more of the base attribute flags defined in &lt;curses.h&gt; on page 220 that have the A_ prefix. The application can extract these components of a ctype value using the base masks defined in &lt;curses.h&gt; for this purpose.</td>
</tr>
</tbody>
</table>
| wchar_t| As described in <stddef.h>.
| cchar_t| A type that can reference a string of wide characters of up to an implementation-dependent length, a colour-pair, and zero or more attributes from the set of all attributes defined in this document. A null cchar_t object is an object that references a empty wide-character string. Arrays of cchar_t objects are terminated by a null cchar_t object. |

SCREEN An opaque terminal representation.

WINDOW An opaque window representation.
3.1 Components

A Curses initialisation function, usually `initscr()`, determines the terminal model in use, by reference to either an argument or an environment variable. If that model is defined in `terminfo`, then the same `terminfo` entry tells Curses exactly how to operate the terminal.

In this case, a comprehensive API lets the application perform terminal operations. The Curses run-time system receives each terminal request and sends appropriate commands to the terminal to achieve the desired effect.

Relationship to the XBD Specification

Applications using Curses should not also control the terminal using capabilities of the general terminal interface defined in the XBD specification, Chapter 9, General Terminal Interface.

There is no requirement that the paradigms that exist while in Curses mode be carried over outside the Curses environment (see `def_prog_mode()` on page 66).

Relationship to Signals

Curses implementations may provide for special handling of the SIGINT, SIGQUIT and SIGTSTP signals if their disposition is SIGDFL at the time `initscr()` is called (see `initscr()` on page 115).

Any special handling for these signals may remain in effect for the life of the process or until the process changes the disposition of the signal.

None of the Curses functions are required to be safe with respect to signals (see `sigaction()` in the XSH specification).

The behaviour of Curses with respect to signals not defined by the XSH specification is unspecified.
3.2 Screens, Windows and Terminals

Screen
A screen is the physical output device of the terminal. In Curses, a SCREEN data type is an opaque data type associated with a terminal. Each window (see below) is associated with a SCREEN.

Windows
The Curses functions permit manipulation of window objects, which can be thought of as two-dimensional arrays of characters and their renditions. A default window called stdscr, which is the size of the terminal screen, is supplied. Others may be created with newwin().

Variables declared as WINDOW * refer to windows (and to subwindows, derived windows, and pads, as described below). These data structures are manipulated with functions described on the reference manual pages in Chapter 6. Among the most basic functions are move() and addch(). More general versions of these functions are included that allow a process to specify a window.

After using functions to manipulate a window, refresh() is called, telling Curses to make the CRT screen look like stdscr.

Line drawing characters may be specified to be output. On input, Curses is also able to translate arrow and function keys that transmit escape sequences into single values. The line drawing characters and input values use names defined in <curses.h>.

Each window has a flag that indicates that the information in the window could differ from the information displayed on the terminal device. Making any change to the contents of the window, moving or modifying the window, or setting the window’s cursor position, sets this flag (touches the window). Refreshing the window clears this flag. (For further information, see is_linetouched() on page 133.)

Subwindows
A subwindow is a window, created within another window (called the parent window), and positioned relative to the parent window. A subwindow can be created by calling derwin(), newpad() or subwin().

Subwindows can be created from a parent window by calling subwin(). The position and size of subwindows on the screen must be identical to or totally within the parent window. Changes to either the parent window or the subwindow affect both. Window clipping is not a property of subwindows.

Ancestors
The term ancestor refers to a window’s parent, or its parent, and so on.
Derived Windows

Derived windows are subwindows whose position is defined by reference to the parent window rather than in absolute screen coordinates. Derived windows are otherwise no different from subwindows.

Pads

A pad is a specialised case of subwindow that is not necessarily associated with a viewable part of a screen. Functions that deal with pads are all discussed in newpad() on page 150.

Terminal

A terminal is the logical input and output device through which character-based applications interact with the user. TERMINAL is an opaque data type associated with a terminal. A TERMINAL data structure primarily contains information about the capabilities of the terminal, as defined by terminfo. A TERMINAL also contains information about the terminal modes and current state for input and output operations. Each screen (see above) is associated with a TERMINAL.
3.3  Characters

3.3.1 Character Storage Size

Historically, a position on the screen has corresponded to a single stored byte. This correspondence is no longer true for several reasons:

• Some characters may occupy several columns when displayed on the screen (see Section 3.3.2).

• Some characters may be non-spacing characters, defined only in association with a spacing character (see Section 3.3.5 on page 17).

• The number of bytes to hold a character from the extended character sets depends on the LC_CTYPE locale category.

The internal storage format of characters and renditions is unspecified. There is no implied correspondence between the internal storage format and the external representation of characters and renditions in objects of type chtype and cchar_t.

3.3.2 Multi-column Characters

Some character sets define multi-column characters that occupy more than one column position when displayed on the screen.

Writing a character whose width is greater than the width of the destination window is an error.

3.3.3 Attributes

Each character can be displayed with attributes such as underlining, reverse video or colour on terminals that support such display enhancements. Current attributes of a window are applied to all characters that are written into the window with waddch(), wadd_wch(), waddstr(), waddchstr(), waddwstr(), waddwchstr() and wprintw(). Attributes can be combined.

Attributes can be specified using constants with the A_ prefix specified in <curses.h>. The A_ constants manipulate attributes in objects of type chtype. Additional attributes can be specified using constants with the WA_ prefix. The WA_ constants manipulate attributes in objects of type attr_t.

Two constants that begin with A_ and WA_ and that represent the same terminal capability refer to the same attribute in the terminfo database and in the window data structure. The effect on a window does not differ depending on whether the application specifies A_ or WA_ constants. For example, when an application updates window attributes using the interfaces that support the A_ values, a query of the window attribute using the function that returns WA_ values reflects this update. When it updates window attributes using the interfaces that support the WA_ values, for which corresponding A_ values exist, a query of the window attribute using the function that returns A_ values reflects this update.
3.3.4 Rendition

The rendition of a character displayed on the screen is its attributes and a colour pair.

The rendition of a character written to the screen becomes a property of the character and moves with the character through any scrolling and insert/delete line/character operations. To the extent possible on a particular terminal, a character's rendition corresponds to the graphic rendition of the character put on the screen.

If a given terminal does not support a rendition that an application program is trying to use, Curses may substitute a different rendition for it.

Colours are always used in pairs (referred to as colour-pairs). A colour-pair consists of a foreground colour (for characters) and a background colour (for the field on which the characters are displayed).

3.3.5 Non-spacing Characters

Some character sets may contain non-spacing characters. (Non-spacing characters are those, other than the '\0' character, for which \texttt{wcwidth()} returns a width of zero.) The application may write non-spacing characters to a window. Every non-spacing character in a window is associated with a spacing character and modifies the spacing character. Non-spacing characters in a window cannot be addressed separately. A non-spacing character is implicitly addressed whenever a Curses operation affects the spacing character with which the non-spacing character is associated.

Non-spacing characters do not support attributes. For interfaces that use wide characters and attributes, the attributes are ignored if the wide character is a non-spacing character. Multi-column characters have a single set of attributes for all columns. The association of non-spacing characters with spacing characters can be controlled by the application using the wide character interfaces. The wide character string functions provide codeset-dependent association.

Two typical effects of a non-spacing character associated with a spacing character called \(c\), are as follows:

- The non-spacing character may modify the appearance of \(c\). (For instance, there may be non-spacing characters that add diacritical marks to characters. However, there may also be spacing characters with built-in diacritical marks.)
- The non-spacing character may bridge \(c\) to the character following \(c\). (Examples of this usage are the formation of ligatures and the conversion of characters into compound display forms, words, or ideograms.)

Implementations may limit the number of non-spacing characters that can be associated with a spacing character, provided any limit is at least 5.

Complex Characters

A complex character is a set of associated characters, which may include a spacing character and may include any non-spacing characters associated with it. A spacing complex character is a spacing character followed by any non-spacing characters associated with it. That is, a spacing complex character is a complex character that includes one spacing character. An example of a code set that has complex characters is ISO/IEC 10646-1:1993.

A complex character can be written to the screen; if it does not include a spacing character, any non-spacing characters are associated with the spacing complex character that exists at the
specified screen position. When the application reads information back from the screen, it obtains spacing complex characters.

The `cchar_t` data type represents a complex character and its rendition. When a `cchar_t` represents a non-spacing complex character (that is, when there is no spacing character within the complex character), then its rendition is not used; when it is written to the screen, it uses the rendition specified by the spacing character already displayed.

An object of type `cchar_t` can be initialised using `setcchar()` and its contents can be extracted using `getcchar()`. The behaviour of functions that take a `cchar_t` input argument is undefined if the application provides a `cchar_t` value that was not initialised in this way or obtained from a Curses function that has a `cchar_t` output argument.

### 3.3.6 Window Properties

Associated with each window are the following properties that affect the placing of characters into the window (see Section 3.4.4 on page 22).

**Window Rendition**

Each window has a rendition, which is combined with the rendition component of the window’s background property described below.

**Window Background**

Each window has a background property. The background property specifies:

- A spacing complex character (the background character) that will be used in a variety of situations where visible information is deleted from the screen.

- A rendition to use in displaying the background character in those situations, and in other situations specified in Section 3.4.4 on page 22.
3.4 Conceptual Operations

3.4.1 Screen Addressing

Many Curses functions use a coordinate pair. In the DESCRIPTION, coordinate locations are represented as \((y, x)\) since the \(y\) argument always precedes the \(x\) argument in the function call. These coordinates denote a line/column position, not a character position.

The coordinate \(y\) always refers to the row (of the window), and \(x\) always refers to the column. The first row and the first column is number 0, not 1. The position \((0, 0)\) is the window's origin.

For example, for terminals that display the ISO 8859-1 character set (with left-to-right writing), \((0, 0)\) represents the upper left-hand corner of the screen.

Functions that start with \texttt{mv} take arguments that specify a \((y, x)\) position and move the cursor (as though \texttt{move()} were called) before performing the requested action. As part of the requested action, further cursor movement may occur, specified on the respective reference manual page.

3.4.2 Basic Character Operations

Adding (Overwriting)

The Curses functions that contain the word add, such as \texttt{addch()}, actually specify one or more characters to replace (overwrite) characters already in the window. If these functions specify only non-spacing characters, they are appended to a spacing character already in the window; see also Section 3.3.5 on page 17.

When replacing a multi-column character with a character that requires fewer columns, the new character is added starting at the specified or implied column position. All columns that the former multi-column character occupied that the new character does not require are orphaned columns, which are filled using the background character and rendition.

Replacing a character with a character that requires more columns also replaces one or more subsequent characters on the line. This process may also produce orphaned columns.

Truncation, Wrapping and Scrolling

If the application specifies a character or a string of characters such that writing them to a window would extend beyond the end of the line (for example, if the application tries to deposit any multi-column character at the last column in a line), the behaviour depends on whether the function supports line wrapping:

- If the function does not wrap, it fails.
- If the function wraps, then it places one or more characters in the window at the start of the next line, beginning with the first character that would not completely fit on the original line.

If the final character on the line is a multi-column character that does not completely fit on the line, the entire character wraps to the next line and columns at the end of the original line may be orphaned.

If the original line was the last line in the window, the wrap may cause a scroll to occur:

- If scrolling is enabled, a scroll occurs. The contents of the first line of the window are lost. The contents of each remaining line in the window move to the previous line. The last line of the window is filled with any characters that wrapped. Any remaining space on the last line is filled with the background character and rendition.
— If scrolling is disabled, any characters that would extend beyond the last column of the last line are truncated.

The `scroll()` function enables and disables scrolling.

Some `add` functions move the cursor just beyond the end of the last character added. If this position is beyond the end of a line, it causes wrapping and scrolling under the conditions specified in the second bullet above.

**Insertion**

Insertion functions (such as `insch()`) insert characters immediately before the character at the specified or implied cursor position.

The insertion shifts all characters that were formerly at or beyond the cursor position on the cursor line toward the end of that line. The disposition of the characters that would thus extend beyond the end of the line depends on whether the function supports wrapping:

- If the function does not wrap, those characters are removed from the window. This may produce orphaned columns.

- If the function supports wrapping, the effect is as described above in *Truncation, Wrapping and Scrolling* on page 19 (except that the overwriting discussed in the final dash is an insertion).

If multi-column characters are displayed, some cursor positions are within a multi-column character but not at the beginning of a character. Any request to insert data at a position that is not the beginning of a multi-column character will be adjusted so that the actual cursor position is at the beginning of the multi-column character in which the requested position occurs.

There are no warning indications relative to cursor relocation. The application should not maintain an image of the cursor position, since this constitutes placing terminal-specific information in the application and defeats the purpose of using Curses.

Portable applications cannot assume that a cursor position specified in an insert function is a reusable indication of the actual cursor position.

**Deletion**

Deletion functions (such as `delch()`) delete the simple or complex character at the specified or implied cursor position, no matter which column of the character this is. All column positions are replaced by the background character and rendition and the cursor is not relocated. If a character-deletion operation would cause a previous wrapping operation to be undone, then the results are unspecified.

**Window Operations**

Overlapping a window (that is, placing one window on top of another) and overwriting a window (that is, copying the contents of one window into another) follows the operation of overwriting multi-column glyphs around its edge. Any orphaned columns are handled as in the character operations.
Characters that Straddle the Subwindow Border

A subwindow can be defined such that multi-column characters straddle the subwindow border. The character operations deal with these straddling characters as follows:

- Reading the subwindow with a function such as `in_wch()` reads the entire straddling character.
- Adding, inserting or deleting in the subwindow deletes the entire straddling character before the requested operation begins and does not relocate the cursor.
- Scrolling lines in the subwindow has the following effects:
  - A straddling character at the start of the line is completely erased before the scroll operation begins.
  - A straddling character at the end of the line moves in the direction of the scroll and continues to straddle the subwindow border. Column positions outside the subwindow at the straddling character's former position are orphaned unless another straddling character scrolls into those positions.

If the application calls a function such as `border()`, the above situations do not occur because writing the border on the subwindow deletes any straddling characters.

In the above cases involving multi-column characters, operations confined to a subwindow can modify the screen outside the subwindow. Therefore, saving a subwindow, performing operations within the subwindow, and then restoring the subwindow may disturb the appearance of the screen. To overcome these effects (for example, for pop-up windows), the application should refresh the entire screen.

3.4.3 Special Characters

Some functions process special characters as specified below.

In functions that do not move the cursor based on the information placed in the window, these special characters would only be used within a string in order to affect the placement of subsequent characters; the cursor movement specified below does not persist in the visible cursor beyond the end of the operation. In functions that do move the cursor, these special characters can be used to affect the placement of subsequent characters and to achieve movement of the visible cursor.

- `<backspace>` Unless the cursor was already in column 0, `<backspace>` moves the cursor one column toward the start of the current line and any characters after the `<backspace>` are added or inserted starting there.

- `<carriage return>`
  Unless the cursor was already in column 0, `<carriage return>` moves the cursor to the start of the current line. Any characters after the `<carriage return>` are added or inserted starting there.

- `<newline>`
  In an add operation, Curses adds the background character into successive columns until reaching the end of the line. Scrolling occurs as described in **Truncation, Wrapping and Scrolling** on page 19. Any characters after the `<newline>` character are added, starting at the start of the new line.

  In an insert operation, `<newline>` erases the remainder of the current line with the background character, effectively a `wclrtoeol()`, and moves the cursor to the start of a new line. When scrolling is enabled, advancing the cursor to a new line may cause scrolling as described in **Truncation, Wrapping and Scrolling** on page 19.
Any characters after the <newline> character are inserted at the start of the new line.

The filter() function may inhibit this processing.

Tab characters in text move subsequent characters to the next horizontal tab stop. By default, tab stops are in column 0, 8, 16, and so on.

In an insert or add operation, Curses inserts or adds, respectively, the background character into successive columns until reaching the next tab stop. If there are no more tab stops in the current line, wrapping and scrolling occur as described in Truncation, Wrapping and Scrolling on page 19.

Control Characters

The Curses functions that perform special-character processing conceptually convert control characters to the caret (’ˆ’) character followed by a second character (which is an upper-case letter if it is alphabetic) and write this string to the window in place of the control character. The functions that retrieve text from the window will not retrieve the original control character.

3.4.4 Rendition of Characters Placed into a Window

When the application adds or inserts characters into a window, the effect is as follows:

If the character is not the space character, then the window receives:

• the character that the application specifies

• the colour that the application specifies; or the window colour, if the application does not specify a colour

• the attributes specified, OR-ed with the window attributes.

If the character is the space character, then the window receives:

• the background character

• the colour that the application specifies; or the window colour, if the application does not specify a colour

• the attributes specified, OR-ed with the window attributes.
3.5 Input Processing

The Curses input model provides a variety of ways to obtain input from the keyboard.

3.5.1 Keypad Processing

The application can enable or disable keypad translation by calling `keypad()` when translation is enabled, Curses attempts to translate a sequence of terminal input that represents the pressing of a function key into a single key code. When translation is disabled, Curses passes terminal input to the application without such translation, and any interpretation of the input as representing the pressing of a keypad key must be done by the application.

The complete set of key codes for keypad keys that Curses can process is specified by the constants defined in `<curses.h>` whose names begin with "KEY_".

Each terminal type described in the terminfo database may support some or all of these key codes. The terminfo database specifies the sequence of input characters from the terminal type that correspond to each key code (see Section A.1.8 on page 262).

The Curses implementation cannot translate keypad keys on terminals where pressing the keys does not transmit a unique sequence.

When translation is enabled and a character that could be the beginning of a function key (such as escape) is received, Curses notes the time and begins accumulating characters. If Curses receives additional characters that represent the pressing of a keypad key, within an unspecified interval from the time the first character was received, then Curses converts this input to a key code for presentation to the application. If such characters are not received during this interval, translation of this input does not occur and the individual characters are presented to the application separately. (Because Curses waits for this interval to accumulate a key code, many terminals experience a delay between the time a user presses the escape key and the time the escape is returned to the application.)

In addition, No Timeout Mode provides that in any case where Curses has received part of a function key sequence, it waits indefinitely for the complete key sequence. The "unspecified interval" in the previous paragraph becomes infinite in No Timeout Mode. No Timeout Mode allows the use of function keys over slow communication lines. No Timeout Mode lets the user type the individual characters of a function key sequence, but also delays application response when the user types a character (not a function key) that begins a function key sequence. For this reason, in No Timeout Mode many terminals will appear to hang between the time a user presses the escape key and the time another key is pressed. No Timeout Mode is switchable by calling `notimeout()`.

If any special characters (see Section 3.4.3 on page 21) are defined or redefined to be characters that are members of a function key sequence, then Curses will be unable to recognise and translate those function keys.

Several of the modes discussed below are described in terms of availability of input. If keypad translation is enabled, then input is not available once Curses has begun receiving a keypad sequence until the sequence is completely received or the interval has elapsed.
3.5.2 Input Mode

The XBD specification (Special Characters) defines flow-control characters, the interrupt character, the erase character, and the kill character. Four mutually-exclusive Curses modes let the application control the effect of these input characters:

<table>
<thead>
<tr>
<th>Input Mode</th>
<th>Effect</th>
</tr>
</thead>
</table>
| Cooked Mode    | This achieves normal line-at-a-time processing with all special characters handled outside the application. This achieves the same effect as canonical-mode input processing as specified in the XBD specification. The state of the ISIG and IXON flags are not changed upon entering this mode by calling `nocbreak()`, and are set upon entering this mode by calling `noraw()`.
|                | The implementation supports erase and kill characters from any supported locale, no matter what the width of the character is. |
| `cbreak` Mode  | Characters typed by the user are immediately available to the application and Curses does not perform special processing on either the erase character or the kill character. An application can select `cbreak` mode to do its own line editing but to let the abort character be used to abort the task. This mode achieves the same effect as non-canonical-mode, Case B input processing (with MIN set to 1 and ICRNL cleared) as specified in the XBD specification. The state of the ISIG and IXON flags are not changed upon entering this mode. |
| Half-Delay Mode| The effect is the same as `cbreak`, except that input functions wait until a character is available or an interval defined by the application elapses, whichever comes first. This mode achieves the same effect as non-canonical-mode, Case C input processing (with TIME set to the value specified by the application) as specified in the XBD specification. The state of the ISIG and IXON flags are not changed upon entering this mode. |
| Raw Mode       | Raw mode gives the application maximum control over terminal input. The application sees each character as it is typed. This achieves the same effect as non-canonical mode, Case D input processing as specified in the XBD specification. The ISIG and IXON flags are cleared upon entering this mode. |

The terminal interface settings are recorded when the process calls `initscr()` or `newterm()` to initialise Curses and restores these settings when `endwin()` is called. The initial input mode for Curses operations is unspecified unless the implementation supports Enhanced Curses compliance, in which the initial input mode is `cbreak` mode.

The behaviour of the BREAK key depends on other bits in the display driver that are not set by Curses.
3.5.3 Delay Mode

Two mutually-exclusive delay modes specify how quickly certain Curses functions return to the application when there is no terminal input waiting when the function is called:

No Delay       The function fails.
Delay          The application waits until the implementation passes text through to the application. If cbreak or Raw Mode is set, this is after one character. Otherwise, this is after the first <newline> character, end-of-line character, or end-of-file character.

The effect of No Delay Mode on function key processing is unspecified.

3.5.4 Echo Processing

Echo mode determines whether Curses echoes typed characters to the screen. The effect of Echo mode is analogous to the effect of the ECHO flag in the local mode field of the termios structure associated with the terminal device connected to the window. However, Curses always clears the ECHO flag when invoked, to inhibit the operating system from performing echoing. The method of echoing characters is not identical to the operating system’s method of echoing characters, because Curses performs additional processing of terminal input.

If in Echo mode, Curses performs its own echoing: Any visible input character is stored in the current or specified window by the input function that the application called, at that window’s cursor position, as though addch() were called, with all consequent effects such as cursor movement and wrapping.

If not in Echo mode, any echoing of input must be performed by the application. Applications often perform their own echoing in a controlled area of the screen, or do not echo at all, so they disable Echo mode.

It may not be possible to turn off echo processing for synchronous and networked asynchronous terminals because echo processing is done directly by the terminals. Applications running on such terminals should be aware that any characters typed will appear on the screen at wherever the cursor is positioned.
3.6 The Set of Curses Functions

The Curses functions allow: overall screen, window and pad manipulation; output to windows and pads; reading terminal input; control over terminal and Curses input and output options; environment query functions; colour manipulation; use of soft label keys; access to the terminfo database of terminal capabilities; and access to low-level functions.

3.6.1 Function Name Conventions

The reference manual pages in Chapter 4 present families of multiple Curses functions. Most function families have different functions that give the programmer the following options:

- A function with the basic name operates on the window stdscr. A function with the same name plus the w prefix operates on a window specified by the win argument.

  When the reference manual page for a function family refers to the current or specified window, it means stdscr for the basic functions and the window specified by win for any w function.

  Functions whose names have the p prefix require an argument that is a pad instead of a window.

- A function with the basic name operates based on the current cursor position (of the current or specified window, as described above). A function with the same name plus the mv prefix moves the cursor to a position specified by the y and x arguments before performing the specified operation.

  When the reference manual page for a function family refers to the current or specified position, it means the cursor position for the basic functions and the position (y, x) for any mv function.

  The mvw prefix exists and combines the mv semantics discussed here with the w semantics discussed above. The window argument is always specified before the coordinates.

- 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 w infix operates on wide (multi-byte) characters. A function with the same name plus the _w infix operates on complex characters and their renditions.

- When a function with the basic name operates on a single character, there is sometimes a function with the same name plus the n infix that operates on multiple characters. An n argument specifies the number of characters to process. The respective manual page specifies the outcome if the value of n is inappropriate.
### 3.6.2 Function Families Provided

<table>
<thead>
<tr>
<th>Function Names</th>
<th>Description</th>
<th>s</th>
<th>w</th>
<th>c</th>
<th>Refer to</th>
</tr>
</thead>
<tbody>
<tr>
<td>[mv][w]addch()</td>
<td>Add (Overwrite)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>addch()</td>
</tr>
<tr>
<td>[mv][w]addch[n]str()</td>
<td>add a character string</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>addchstr()</td>
</tr>
<tr>
<td>[mv][w]add[n]str()</td>
<td>add a string</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>addnstr()</td>
</tr>
<tr>
<td>[mv][w]add[n]wstr()</td>
<td>add a wide character string</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>addnwstr()</td>
</tr>
<tr>
<td>[mv][w]add_wch()</td>
<td>add an array of wide characters and renditions</td>
<td>?</td>
<td>N</td>
<td>N</td>
<td>add_wchnstr()</td>
</tr>
<tr>
<td>[mv][w]chgat()</td>
<td>change renditions of characters in a window</td>
<td>-</td>
<td>N</td>
<td>N</td>
<td>chgat()</td>
</tr>
<tr>
<td>[mv][w]delch()</td>
<td>delete a character</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>delch()</td>
</tr>
<tr>
<td>[mv][w]getch()</td>
<td>Get (Input from Keyboard to Window)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>getch()</td>
</tr>
<tr>
<td>[mv][w]get[n]str()</td>
<td>get a character string</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>getnstr()</td>
</tr>
<tr>
<td>[mv][w]get_wch()</td>
<td>get a wide character string</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>get_wch()</td>
</tr>
<tr>
<td>[mv][w]get_wstr()</td>
<td>get an array of wide characters and key codes</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>get_wstr()</td>
</tr>
<tr>
<td>[w]move()</td>
<td>move the cursor</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>move()</td>
</tr>
<tr>
<td>[mv][w]inch()</td>
<td>Input (Read Back from Window)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>inch()</td>
</tr>
<tr>
<td>[mv][w]inch[n]str()</td>
<td>input a string of characters and attributes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>inchnstr()</td>
</tr>
<tr>
<td>[mv][w]in[n]str()</td>
<td>input a string</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>instr()</td>
</tr>
<tr>
<td>[mv][w]in[n]wstr()</td>
<td>input a string of wide characters</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>innwstr()</td>
</tr>
<tr>
<td>[mv][w]in_wch()</td>
<td>input a wide character and rendition</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>in_wch()</td>
</tr>
<tr>
<td>[mv][w]in_wch[n]str()</td>
<td>input an array of wide characters and renditions</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>inwchnstr()</td>
</tr>
<tr>
<td>[mv][w]insch()</td>
<td>insert a character</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>insch()</td>
</tr>
<tr>
<td>[mv][w]ins[n]str()</td>
<td>insert a character string</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>insnstr()</td>
</tr>
<tr>
<td>[mv][w]ins[n]wstr()</td>
<td>insert a wide-character string</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>insnwstr()</td>
</tr>
<tr>
<td>[mv][w]ins_wch()</td>
<td>insert a wide character</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>ins_wch()</td>
</tr>
</tbody>
</table>

**Legend**

The following notation indicates the effect when characters are moved to the screen. (For the Get functions, this applies only when echoing is enabled.)

- **s** Y means these functions perform special-character processing (see Section 3.4.3 on page 21). N means they do not. ? means the results are unspecified when these functions are applied to special characters.

- **w** Y means these functions perform wrapping (see Truncation, Wrapping and Scrolling on page 19). N means they do not.

- **c** Y means these functions advance the cursor (see Truncation, Wrapping and Scrolling on page 19). N means they do not.
- The attribute specified by this column does not apply to these functions.
3.7 Interfaces Implemented as Macros

The following interfaces with arguments must be implemented as macros. The relevance to the application programmer is that the ‘&’ character cannot be used before the arguments.

<table>
<thead>
<tr>
<th>Macros</th>
<th>Chapter 4 Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>getbegyx(), getmaxyx(), getparyx(), getyx()</td>
<td>getbegyx()</td>
</tr>
</tbody>
</table>

The header file reference manual pages list other macros, like COLOR_BLACK, that do not take arguments.

3.8 Initialised Curses Environment

Before executing an application that uses Curses, the terminal must be prepared as follows:

- If the terminal has hardware tab stops, they should be set.
- Any initialisation strings defined for the terminal must be output to the terminal.

The resulting state of the terminal must be compatible with the model of the terminal that Curses has, as reflected in the terminal’s entry in the `terminfo` database (see Chapter 6).

To initialise Curses, the application must call `initscr()` or `newterm()` before calling any of the other functions that deal with windows and screens, and it must call `endwin()` before exiting. To get character-at-a-time input without echoing (most interactive, screen-oriented programs want this), the following sequence should be used:

```c
initscr()
cbreak()
noecho()
```

Most programs would additionally use the sequence:

```c
nonl()
intrflush(stdscr, FALSE)
keypad(stdscr, TRUE)
```
3.9 Synchronous and Networked Asynchronous Terminals

This section indicates to the application writer some considerations to be borne in mind when driving synchronous, networked asynchronous (NWA) or non-standard directly-connected asynchronous terminals.

Such terminals are often used in a mainframe environment and communicate to the host in block mode. That is, the user types characters at the terminal then presses a special key to initiate transmission of the characters to the host.

Frequently, although it may be possible to send arbitrary sized blocks to the host, it is not possible or desirable to cause a character to be transmitted with only a single keystroke. This can cause severe problems to an application wishing to make use of single-character input; see Section 3.5 on page 23.

Output

The Curses interface can be used in the normal way for all operations pertaining to output to the terminal, with the possible exception that on some terminals the refresh() routine may have to redraw the entire screen contents in order to perform any update.

If it is additionally necessary to clear the screen before each such operation, the result could be undesirable.

Input

Because of the nature of operation of synchronous (block-mode) and NWA terminals, it might not be possible to support all or any of the Curses input functions. In particular, the following points should be noted:

- Single-character input might not be possible. It may be necessary to press a special key to cause all characters typed at the terminal to be transmitted to the host.

- It is sometimes not possible to disable echo. Character echo may be performed directly by the terminal. On terminals that behave in this way, any Curses application that performs input should be aware that any characters typed will appear on the screen at wherever the cursor is positioned. This does not necessarily correspond to the position of the cursor in the window.
This chapter describes the Curses functions, macros and external variables to support application portability at the C-language source level.

The display model defined in Section 3.4 on page 19 contains important information, not repeated for individual interface definitions, regarding cursor movement, relocation of the cursor in the case of multi-column characters, wrapping of characters to subsequent lines of the screen, truncation of characters, and other important concepts. The reference manual pages must be read in conjunction with this overview information.

The interface definitions are collated as though any underscore characters were not present.
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
Section 3.4.4 on page 22, add_wch(), attroff(), doupdate(), <curses.h>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity. Also the type of argument ch is changed from chtype to const
chtype.
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 chtype 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(), mvaddchstr(), mvwaddchstr() and waddchstr() functions copy at most n items, but no more than will fit on the current or specified line. If n is −1 then the whole string is copied, to the maximum number that fit on the current or specified 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>.

CHANGE HISTORY
First released in Issue 4.
addnstr() ENHANCED CURSES Curses Interfaces

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, const char *str, int n);
int mvwaddstr(WINDOW *win, int y, int x, const char *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() (in the XSH specification), <curses.h>.

CHANGE HISTORY

First released in Issue 4.

In Issue 3, the addstr(), mvaddstr(), mvwaddstr() and waddstr() functions were described in the addstr() entry. In Issue 4, the type of the str argument defined for these functions is changed from char * to const char *, and the DESCRIPTION is changed to indicate that the functions will handle multi-byte sequences correctly.


34 X/Open CAE Specification 1996
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 \textit{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 \textit{cchar_t} from the \textit{wchar_t} and the background rendition and calling \textit{wadd_wch()}, once for each \textit{wchar_t} character in the string. The cursor movement specified by the \textit{mv} functions occurs only once at the start of the operation.

The \texttt{addnwstr()}, \texttt{mvaddnwstr()}, \texttt{mvwaddnwstr()} and \texttt{waddnwstr()} functions write at most \textit{n} wide characters. If \textit{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
\texttt{add_wch()}, \texttt{<curses.h>}

CHANGE HISTORY
First released in Issue 4.

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(const cchar_t * wch);
int wadd_wch(WINDOW * win, const cchar_t * wch);
int mvadd_wch(int y, int x, const cchar_t * wch);
int mvwadd_wch(WINDOW * win, int y, int x, const cchar_t * 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 \texttt{wch} refers to a spacing character, then any previous character at that location is removed, a new character specified by \texttt{wch} is placed at that location with rendition specified by \texttt{wch}; then the cursor advances to the next spacing character on the screen.

- If \texttt{wch} refers to a non-spacing character, all previous characters at that location are preserved, the non-spacing characters of \texttt{wch} are added to the spacing complex character, and the rendition specified by \texttt{wch} is ignored.

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

ERRORS
No errors are defined.

SEE ALSO
Section 3.4.4 on page 22, \addch(), \texttt{<curses.h>}

CHANGE HISTORY
First released in Issue 4.

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

SYNOPSIS

```c
#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() functions end successfully after writing n cchar_t's (or the entire array of cchar_t's, if n is −1).

RETURN VALUE

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

ERRORS

No errors are defined.

SEE ALSO

<cursh.h>.

CHANGE HISTORY

First released in Issue 4.

ATTRON, ATTROFF, ATTRSET, WTROFF, WATTRON, WATTRSET — restricted window attribute control functions

SYNOPSIS

#include <curses.h>

int attroff(int attr);
int attroff(WINDOW *win, int attr);
int attset(int attr);
int attset(WINDOW *win, int attr);
int wattron(int attr);
int wattron(WINDOW *win, int attr);
int wattroff(int attr);
int wattroff(WINDOW *win, int attr);

DESCRIPTION

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

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

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

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>.

CHANGE HISTORY

First released in Issue 2.

Issue 4

This entry is rewritten for clarity. The DESCRIPTION is updated to specify that it is undefined whether these functions can be used to manipulate attributes beyond those defined in Issue 3. The standend(), standout(), wstandend() and wstandout() functions are moved to the standend() entry.
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);
int wattr_get(WINDOW *win, attr_t *attrs, short *color_pair_number, void *opts);
int wattr_off(WINDOW *win, attr_t attrs, void *opts);
int wattr_on(WINDOW *win, attr_t attrs, void *opts);
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 colour 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 colour of the current or specified
window to color_pair_number.

RETURN VALUE
These functions always return OK.

ERRORS
No errors are defined.

SEE ALSO
attroff(), <curses.h>.

CHANGE HISTORY
First released in Issue 4
**Issue 4, Version 2**

This entry is rewritten to include the colour handling functions `wcolor_set()` and `color_set()`. 
NAME
   baudrate — get terminal baud rate

SYNOPSIS
   #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() (in the XSH specification), <curses.h>.

CHANGE HISTORY
   First released in Issue 2.

Issue 4
   The argument list is explicitly declared as void.
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>`.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The argument list is explicitly declared as `void`. The RETURN VALUE section is changed to indicate that the function always returns OK. The `flash()` function is moved to its own entry.
NAME

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

SYNOPSIS

```c
#include <curses.h>

int bkgd(chtype ch);
void bkgdset(chtype ch);
chttype 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, logical OR the requested attributes into the window rendition, and set the background property of the current or specified window based on the information in `ch`. If `ch` refers to a multi-column character, the results are undefined.

The `bkgd()` and `wbkgd()` functions turn off the previous background attributes, logical 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 window rendition.
- Wherever the former background character appears, it is changed to the new background character.

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 (`chttype`)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

Section 3.3.4 on page 17, `<curses.h>`.

CHANGE HISTORY

First released in Issue 4.

Issue 4, Version 2
Rewritten for clarity.
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 window 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

Section 3.3.4 on page 17, `<curses.h>`.

CHANGE HISTORY

First released in Issue 4.

NAME

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

SYNOPSIS

```c
#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,
            chtype 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>`.

CHANGE HISTORY

First released in Issue 4.
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>`.

CHANGE HISTORY
First released in Issue 4.
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>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The DESCRIPTION is changed to describe this function in terms of a call to the wborder() function.
NAME
box_set — draw borders from complex characters and renditions

SYNOPSIS

```c
#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:

```c
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>`.

CHANGE HISTORY
First released in Issue 4.

NAME
can_change_color, color_content, has_colors, init_color, init_pair, start_color, pair_content —
colour 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 colour on terminals that support colour.

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

Initialisation
The start_color() function must be called in order to enable use of colours and before any colour manipulation function is called. The function initialises eight basic colours (black, blue, green, cyan, red, magenta, yellow, and white) that can be specified by the colour macros (such as COLOR_BLACK) defined in <curses.h>. (See Colour-related Macros on page 223.) The initial appearance of these eight colours is not specified.

The function also initialises two global external variables:

• COLORS defines the number of colours that the terminal supports. (See Colour Identification below.) If COLORS is 0, the terminal does not support redefinition of colours (and can_change_color() will return FALSE).

• COLOR_PAIRS defines the maximum number of colour-pairs that the terminal supports. (See User-defined Colour Pairs below.)

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

The \texttt{init\_color()} function redefines colour number \texttt{color}, on terminals that support the redefinition of colours, to have the red, green, and blue intensity components specified by \texttt{red}, \texttt{green}, and \texttt{blue}, respectively. Calling \texttt{init\_color()} also changes all occurrences of the specified colour on the screen to the new definition.

The \texttt{color\_content()} function identifies the intensity components of colour number \texttt{color}. It stores the red, green, and blue intensity components of this colour in the addresses pointed to by \texttt{red}, \texttt{green}, and \texttt{blue}, respectively.

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

User-Defined Colour Pairs

Calling \texttt{init\_pair()} defines or redefines colour-pair number \texttt{pair} to have foreground colour \texttt{f} and background colour \texttt{b}. Calling \texttt{init\_pair()} changes any characters that were displayed in the colour pair’s old definition to the new definition and refreshes the screen.

After defining the colour pair, the macro \texttt{COLOR\_PAIR(n)} returns the value of colour pair \texttt{n}. This value is the colour attribute as it would be extracted from a \texttt{chtype}. Conversely, the macro \texttt{PAIR\_NUMBER(value)} returns the colour pair number associated with the colour attribute \texttt{value}.

The \texttt{pair\_content()} function retrieves the component colours of a colour-pair number \texttt{pair}. It stores the foreground and background colour numbers in the variables pointed to by \texttt{f} and \texttt{b}, respectively.

With \texttt{init\_pair()} and \texttt{pair\_content()}, the value of \texttt{pair} must be in a range from 0 to and including \texttt{COLOR\_PAIRS}–1. (There may be an implementation-specific upper limit on the valid value of \texttt{pair}, but any such limit is at least 63.) Valid values for \texttt{f} and \texttt{b} are the range from 0 to and including \texttt{COLORS}–1.

RETURN VALUE

The \texttt{has\_colors()} function returns TRUE if the terminal can manipulate colors; otherwise, it returns FALSE.

The \texttt{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, \texttt{start\_color()} must be called, usually right after \texttt{initscr()}.

The \texttt{can\_change\_color()} and \texttt{has\_colors()} functions facilitate writing terminal-independent programs. For example, a programmer can use them to decide whether to use colour or some other video attribute.

On colour terminals, a typical value of \texttt{COLORS} is 8 and the macros such as \texttt{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

\texttt{attroff()}, \texttt{delscreen()}, \texttt{<curses.h>}. 
CHANGE HISTORY

First released in Issue 4.

Corrections made in section "NAME" and section "APPLICATION USAGE", Issue 4, Version 2.
NAME
cbreak, nocbreak, noraw, raw — input mode control functions

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

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
Section 3.5.2 on page 24, <curses.h>, XBD specification, Chapter 9, General Terminal Interface.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The raw() and noraw() functions are merged with this entry. In previous issues, they appeared
in entries of their own.
The entry is rewritten for clarity. The argument list for all these functions is explicitly declared as void.
NAME
chgat, mvchgat, mvwchgat, wchgat — change renditions of characters in a window

SYNOPSIS

```c
#include <curses.h>

int chgat(int n, attr_t attr, short color, const void *opts);
int mvchgat(int y, int x, int n, attr_t attr, short color,
            const void *opts);
int mvwchgat(WINDOW *win, int y, int x, int n, attr_t attr,
            short color, const void *opts);
int wchgat(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 current or specified line, if \( n \) is \( -1 \)), starting at the current or specified cursor position. The attributes and colors are specified by \( \text{attr} \) and \( \text{color} \) 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 \( \text{opts} \) argument is reserved for definition in a future edition of this document. Currently, the application must provide a null pointer as \( \text{opts} \).

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

ERRORS
No errors are defined.

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

CHANGE HISTORY
First released in Issue 4.
NAME

clear, erase, wclear, werase — clear a window

SYNOPSIS

#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>.

CHANGE HISTORY

First released in Issue 2.

Issue 4

The erase() and werase() functions are merged with this entry. In previous issues, they appeared in entries of their own.

The entry is rewritten for clarity. The argument list for the clear() and erase() functions is explicitly declared as void.
NAME

clearok, idlok, leaveok, scrollok, setscrreg, wsetsrcreg — 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 wsetsrcreg(WINDOW *win, int top, int bot);

DESCRIPTION

These functions set options that deal with output within Curses.

The clearok() function assigns the value of 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 bf is TRUE, use of these features is enabled. If 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 bf is TRUE, refresh operations on the specified window may leave the terminal’s cursor at an arbitrary position. If 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 bf is TRUE, then scrolling is enabled for the specified window, with the consequences discussed in Truncation, Wrapping and Scrolling on page 19. If bf is FALSE, scrolling is disabled for the specified window. The initial state is FALSE.

The setscrreg() and wsetsrcreg() functions define a software scrolling region in the current or specified window. The top and 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 wsetsrcreg() 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
clearok() visually annoying.

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

SEE ALSO

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

CHANGE HISTORY

First released in Issue 2.

Issue 4

The idlok(), leaveok(), scrollok(), setscreg() and wsetscreg() functions are merged with this entry. In previous issues, they appeared in entries of their own.

The entry is rewritten for clarity. The DESCRIPTION of clearok() is updated to indicate that clearing of a screen applies if the flag is TRUE in either curscr or the specified window.

The RETURN VALUE is changed to indicate that the clearok(), leaveok() and scrollok() functions always return OK.
NAME
clrtobot, wclrtobot — clear from cursor to end of window

SYNOPSIS
#include <curses.h>
int clrtobot(void);
int wclrtobot(WINDOW *win);

DESCRIPTION
The clrtobot() and wclrtobot() functions erase all lines following the cursor in the current or specified window, and erase the current line from the cursor to the end of the line, inclusive. These functions do not update the cursor.

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

ERRORS
No errors are defined.

SEE ALSO
doupdate(), <curses.h>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity. The argument list for the clrtobot() function is explicitly declared as void.
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. These functions do not update the cursor.

RETURN VALUE

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

ERRORS

No errors are defined.

SEE ALSO

doupdate(), <curses.h>.

CHANGE HISTORY

First released in Issue 2.

Issue 4

The entry is rewritten for clarity. The argument list for the clrtoeol() function is explicitly declared as void.
NAME
color_content — identify red, green and blue intensity of a colour

SYNOPSIS
EC
#include <curses.h>

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

DESCRIPTION
Refer to \textit{can\_change\_color}().

CHANGE HISTORY
First released in Issue 4.
NAME
COLOR_PAIRS, COLORS — external variables for colour support

SYNOPSIS
#include <curses.h>
extern int COLOR_PAIRS;
extern int COLORS;

DESCRIPTION
Refer to can_change_color().

CHANGE HISTORY
First released in Issue 4.
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>.

CHANGE HISTORY
First released in Issue 4.
copywin() — copy a region of a window

SYNOPSIS

```c
#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>`.

CHANGE HISTORY

First released in Issue 4.

NAME
curscr — current window

SYNOPSIS
\begin{verbatim}
#include <curses.h>
extern WINDOW *curscr;
\end{verbatim}

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>.

CHANGE HISTORY
First released in Issue 4.
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>.

CHANGE HISTORY
First released in Issue 4.
NAME

cur_term — current terminal information

SYNOPSIS

```c
#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>`.

CHANGE HISTORY

First released in Issue 4.
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>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The reset_prog_mode() and reset_shell_mode() functions are merged with this entry. In previous issues, they appeared in entries of their own.

The entry is rewritten for clarity. The argument list for all these functions is explicitly declared as void.
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
Section 6.1.3 on page 241, napms(), <curses.h>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity.
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
  <curses.h>.

CHANGE HISTORY
  First released in Issue 2.

Issue 4
  The entry is rewritten for clarity. The argument list for the delch() function is explicitly declared as void.
NAME
del_curterm, restartterm, set_curterm, setupterm — interfaces to the terminfo database

SYNOPSIS

```c
#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);
```

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 initialises 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:

```c
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
   Section A.3 on page 279, baudrate(), erasechar(), has_ic(), longname(), putc(), termattrs(), termname(), tgetent(), tigetflag(), use_env(), <term.h>.

CHANGE HISTORY
   First released in Issue 4.
NAME
deleteln, wdeleteln — delete lines in a window

SYNOPSIS
#include <curses.h>
int deleteln(void);
int wdeleteln(WINDOW *win);

DESCRIPTION
The deleteln() and wdeleteln() 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
insdeln(), <curses.h>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity. The argument list for the deleteln() function is explicitly
declared as void.
delscreen()  

NAME  
delscreen — free storage associated with a screen

SYNOPSIS  
```c  
#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>.

CHANGE HISTORY  
First released in Issue 4.
NAME
delwin — delete a window

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

DESCRIPTION
The delwin() function deletes 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
derwin(), dupwin(), <curses.h>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity.
NAME

derwin, newwin, subwin — window creation functions

SYNOPSIS

```c
#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 `begin_y` and `begin_x` are relative to the origin of the window `orig` rather than absolute screen positions.

The `newwin()` function creates a new window with `nlines` lines and `ncols` columns, positioned so that the origin is `(begin_y, begin_x)`. If `nlines` is zero, it defaults to `LINES - begin_y`; if `ncols` is zero, it defaults to `COLS - begin_x`.

The `subwin()` function creates a new window with `nlines` lines and `ncols` columns, positioned so that the origin is at `(begin_y, begin_x)`. (This position is an absolute screen position, not a position relative to the window `orig`.) If any part of the new window is outside `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>.

CHANGE HISTORY

First released in Issue 4.
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>.

CHANGE HISTORY
First released in Issue 4.
This entry is a merger of the Issue 3 entries refresh() and wnoutrefresh(). The DESCRIPTION is
rewritten for clarity and the argument list for the doupdate() and refresh() functions is explicitly
declared as void. Otherwise the functionality is identical to that defined in Issue 3.
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>.

CHANGE HISTORY
First released in Issue 4.
NAME
echo, noecho — enable/disable terminal echo

SYNOPSIS
#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 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 behaviour is
undefined.

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

ERRORS
No errors are defined.

SEE ALSO
Section 3.5 on page 23, getch(), <curses.h>, XBD specification, Section 9.2, Parameters That Can
Be Set.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity. The argument list for the echo() and noecho() functions is
explicitly declared as void.

Issue 4, Version 2
Further clarification of the state of the echo modes.
NAME
echochar, wechochar — echo single-byte character and rendition to a window and refresh

SYNOPSIS
#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(), doupdate(), echo_wchar(), <curses.h>.

CHANGE HISTORY
First released in Issue 4.
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()`, `doupdate()`, `<curses.h>`.

CHANGE HISTORY

First released in Issue 4.

endwin() CURSES Curses Interfaces

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>.

CHANGE HISTORY
   First released in Issue 2.

   Issue 4
   The entry is rewritten for clarity. The argument list is explicitly declared as void.
NAME
erase, werase — clear a window

SYNOPSIS
#include <curses.h>
int erase(void);
int werase(WINDOW *win);

DESCRIPTION
Refer to clear().

CHANGE HISTORY
First released in Issue 2.

Issue 4
The functionality previously described by this entry is moved to the entry for clear().
NAME
erasechar, erasewchar, killchar, killwchar — terminal environment query functions

SYNOPSIS
#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
Section 3.3.3 on page 16, clearok(), delscreen(), tcgetattr() (in the XSH specification), <curses.h>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The killchar() function is merged with this entry. In previous issues, it appeared in an entry of its own.

The entry is rewritten for clarity. The argument list for the erasechar() and killchar() functions is explicitly declared as void. The functions erasewchar() and killwchar() are added and marked as an X/Open UNIX Extension.
NAME

filter — disable use of certain terminal capabilities

SYNOPSIS

```c
#include <curses.h>

void filter(void);
```

DESCRIPTION

The `filter()` function changes the algorithm for initialising 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

Section 6.1.3 on page 241, `initscr()`, `<curses.h>`.

CHANGE HISTORY

First released in Issue 4.
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>.

CHANGE HISTORY
First released in Issue 4.

In previous issues, this function was included in the entry for beep(). It is moved to its own entry in Issue 4, the argument list is explicitly declared as void, and the RETURN VALUE section is changed to indicate that the function always returns OK.
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>.

CHANGE HISTORY
  First released in Issue 2.

Issue 4
  The entry is rewritten for clarity. The argument list for the flushinp() function is explicitly declared as void.
NAME
getbegyx, 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)  \
{  \
    int _y;
    \
    getbegyx(_win,_y,_x);
}  \

SEE ALSO
<curses.h>.
CHANGE HISTORY

First released in Issue 4.

Corrections made to "APPLICATION USAGE" section, Issue 4, Version 2.
NAME
getbkgd — get background character and rendition using a single-byte character

SYNOPSIS
EC

```c
#include <curses.h>

cttype getbkgd(WINDOW *win);
```

DESCRIPTION
Refer to `bkgd()`.

CHANGE HISTORY
First released in Issue 4.
NAME
getbkgrnd — get background character and rendition

SYNOPSIS
#include <curses.h>

int getbkgrnd(cchar_t *ch);

DESCRIPTION
Refer to bkgrnd().

CHANGE HISTORY
First released in Issue 4.
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 colour
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>.

CHANGE HISTORY
First released in Issue 4.
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>.

Processing of terminal input is subject to the general rules described in Section 3.5 on page 23.

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>, <left-arrow>
and the current erase character:
The input is interpreted as specified in Section 3.4.3 on page 21
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
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
Section 3.5 on page 23, cbreak(), douupdate(), insch(), <curses.h>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity. The argument list for the getch() function is explicitly declared
getch() as void.

Issue 4, Version 2
The Return Value section is expanded.
NAME
getmaxyx — get size of a window

SYNOPSIS

```c
#include <curses.h>

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

DESCRIPTION
Refer to `getbegyx()`.

CHANGE HISTORY
First released in Issue 4.
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, carriage
return or end-of-file 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 mvgetnstr() 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 getnstr() 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

Section 3.5 on page 23, beep(), getch(), <curses.h>.

CHANGE HISTORY

First released in Issue 4.
In Issue 3, the `getstr()`, `mvgetstr()`, `mvwgetstr()` and `wgetstr()` functions were described in the `addstr()` entry. In Issue 4, the DESCRIPTION of these functions is rewritten for clarity and is updated to indicate that they will handle multi-byte sequences correctly.

Corrections made to first sentence of "DESCRIPTION", Issue 4, Version 2.
NAME

getn_wstr, get_wstr, mvgetn_wstr, mvget_wstr, mvwgetn_wstr, mvwget_wstr, wgetn_wstr, wget_wstr — get an array of wide characters and function key code from a terminal

SYNOPSIS

```c
#include <curses.h>

int getn_wstr(wchar_t *wstr, int n);
int get_wstr(wchar_t *wstr);
int mvgetn_wstr(int y, int x, wchar_t *wstr, int n);
int mvget_wstr(int y, int x, wchar_t *wstr);
int mvwgetn_wstr(WINDOW *win, int y, int x, wchar_t *wstr, int n);
int mvwget_wstr(WINDOW *win, int y, int x, wchar_t *wstr);
int wgetn_wstr(WINDOW *win, wchar_t *wstr, int n);
int wget_wstr(WINDOW *win, wchar_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 `wmovew()` and then a series of calls to `wget_wch()` were made. The effect of `mvwget_nwstr()` is as though a call to `wmovew()` 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 `get_wstr()`, `mvget_wstr()`, `mvwget_wstr()` or `wget_wstr()` 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>` (in the XSH specification), XBD specification, Chapter 9, General Terminal Interface.

X/Open CAE Specification 1996
CHANGE HISTORY
First released in Issue 4.
NAME
getparyx — get subwindow origin coordinates

SYNOPSIS
#include <curses.h>

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

DESCRIPTION
Refer to getbegyx().

CHANGE HISTORY
First released in Issue 4.
NAME
getstr — get a multi-byte character string from the terminal

SYNOPSIS
#include <curses.h>

int getstr(char *str);

DESCRIPTION
Refer to getnstr().

CHANGE HISTORY
First released in Issue 2.

Issue 4
The functionality previously described by this entry is moved to the entry for getnstr().
get_wch()  ENHANCED CURSES  Curses Interfaces

NAME
get_wch, mvget_wch, mvwget_wch, wget_wch — get a wide character from a terminal

SYNOPSIS
```c
#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 described in Section 3.5 on page 23.

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 as specified in Section 3.4.3 on page 21 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
Section 3.5 on page 23, beep(), cbreak(), ins_wch(), keypad(), move(), <curses.h>, <wchar.h> (in the XSH specification).

CHANGE HISTORY
First released in Issue 4.
NAME
getwin, putwin — dump window to, and reload window from, a file

SYNOPSIS
```
#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 initialises a new window using that data.

The putwin() function writes all data associated with win into thestdio 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>.

CHANGE HISTORY
First released in Issue 4.
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().

CHANGE HISTORY
First released in Issue 4.
NAME
   getyx — get cursor coordinates

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

DESCRIPTION
   Refer to 

CHANGE HISTORY
   First released in Issue 2.

Issue 4
   The functionality previously described by this entry is moved to the entry for getbegyx().
NAME
halfdelay — control input character delay mode

SYNOPSIS

```c
#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` 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
Section 3.5.2 on page 24, `cbreak()`, `<curses.h>` XBD specification, Chapter 9, General Terminal Interface.

CHANGE HISTORY
First released in Issue 4.
NAME
has_colors — indicate whether terminal supports colours

SYNOPSIS

```
#include <curses.h>
bool has_colors(void);
```

DESCRIPTION
Refer to `can_change_color()`.

CHANGE HISTORY
First released in Issue 4.
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>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The has_il() function is merged with this entry. In previous issues, it appeared in an entry of its own.
The entry is rewritten for clarity. The argument list for the has_ic() and has_il() functions is explicitly declared as void.
NAME
hline, mvhline, mvvline, mvwhline, mvwvline, vline, whline, wvline — draw lines from single-byte characters and renditions

SYNOPSIS

```c
#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 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(), 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.

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

CHANGE HISTORY
First released in Issue 4.
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, const cchar_t *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()`, `mvwhline_set()` and `whline_set()` functions draw a line proceeding toward the last column of the same line.

The `vline_set()`, `mvvline_set()`, `mvwvline_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()`, `<curses.h>`.

CHANGE HISTORY

First released in Issue 4.

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>.

CHANGE HISTORY
First released in Issue 4.
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().

CHANGE HISTORY
First released in Issue 2.

Issue 4
The functionality previously described by this entry is moved to the entry for clearok().
NAME
immedok — enable or disable immediate terminal refresh

SYNOPSIS

```c
#include <curses.h>

void immedok(WINDOW *win, bool bf);
```

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>`.

CHANGE HISTORY
First released in Issue 4.
NAME
inch, mvinch, mvwinch, winch — input a single-byte character and rendition from a window

SYNOPSIS
#include <curses.h>
chttype inch(void);
chttype mvinch(int y, int x);
chttype mvwinch(WINDOW *win, int y, int x);
chttype 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 (chttype)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>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity. The argument list for the inch() function is explicitly declared as void.
NAME

inchnstr, inchstr, mvwinchnstr, mvwinchstr, winchnstr, winchstr — input an array of single-byte characters and renditions from a window

SYNOPSIS

```
#include <curses.h>

int inchnstr(chtype *chstr, int n);
int inchstr(chtype *chstr);
int mvwinchnstr(int y, int x, chtype *chstr, int n);
int mvwinchstr(int y, int x, chtype *chstr);
int mvwinchstr(WINDOW *win, int y, int x, chtype *chstr);
int mvwinchstr(WINDOW *win, int y, int x, chtype *chstr);
int winchnstr(WINDOW *win, chtype *chstr, int n);
int winchstr(WINDOW *win, chtype *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()`, `mvwinchnstr()`, `mvwinchstr()` and `winchstr()` 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()`, `mvwinchstr()`, `mvwinchstr()` or `winchstr()` causes undefined results. The use of `inchnstr()`, `mvwinchnstr()`, `mvwinchstr()` or `winchstr()`, respectively, is recommended.

SEE ALSO

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

CHANGE HISTORY

First released in Issue 4.
NAME
init_color, init_pair — redefine specified colour or colour 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().

CHANGE HISTORY
First released in Issue 4.
NAME

initscr, newterm — screen initialisation 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 initialises 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 initialise the `cur_term` external variable.

RETURN VALUE

Upon successful completion, `initscr()` returns a pointer to `stdscr`. Otherwise, it does not return.

Upon successful completion, `newterm()` returns a pointer to the specified terminal. Otherwise, it returns a null pointer.

ERRORS

No errors are defined.

APPLICATION USAGE

A program that outputs to more than one terminal should use `newterm()` for each terminal instead of `initscr()`. A program that needs an indication of error conditions, so it can continue to run in a line-oriented mode if the terminal cannot support a screen-oriented program, would also use this function.

Applications should perform any required handling of the SIGINT, SIGQUIT or SIGTSTP signals before calling `initscr()`.

SEE ALSO

Section A.3 on page 279, `delscreen()`, `doupdate()`, `del_curterm()`, `filter()`, `slk_atroff()`, `use_env()`, `<curses.h>`.

CHANGE HISTORY

First released in Issue 2.
**Issue 4**

The `newterm()` function is merged with this entry. In previous issues, it appeared in an entry of its own.

The entry is rewritten for clarity. The argument list for the `initscr()` function is explicitly declared as `void`. 
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(), mvwinstr() and winstr() 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>.

CHANGE HISTORY
First released in Issue 4.
NAME

innwstr, inwstr, mvinnwstr, mvinwstr, mvwinnwstr, mvwinwstr, winwstr, winnstr — input a string of wide characters from a window

SYNOPSIS

```c
#include <curses.h>

int innwstr(wchar_t *wstr, int n);
int inwstr(wchar_t *wstr);
int mvinnwstr(int y, int x, wchar_t *wstr, int n);
int mvinwstr(int y, int x, wchar_t *wstr);
int mvwinnwstr(WINDOW *win, int y, int x, wchar_t *wstr, int n);
int mvwinwstr(WINDOW *win, int y, int x, wchar_t *wstr);
int winnwstr(WINDOW *win, wchar_t *wstr, int n);
int winwstr(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, inwstr(), 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 inwstr(), mvinwstr(), mvwinnwstr() or winnstr() causes undefined results. The use of innwstr(), mvinnwstr(), mvwinnwstr() or winnstr(), respectively, is recommended.

These functions do not return rendition information.

SEE ALSO

<curses.h>.

CHANGE HISTORY

First released in Issue 4.
NAME
insch, mvinsch, mvwinsch, winsch — insert a single-byte character and rendition into a window

SYNOPSIS
#include <curses.h>
int insch(chtype ch);
int mvinsch(int y, int x, chtype ch);
int mvwinsch(WINDOW * win, int y, int x, chtype ch);
int winsch(WINDOW * win, chtype ch);

DESCRIPTION
These functions insert the character and rendition from 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 A_ prefix.

SEE ALSO
ins_wch(), <curses.h>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity.
Further clarification in the "DESCRIPTION" section, Issue 4, Version 2.
NAME
insdelln, winsdelln — delete or insert lines into a window

SYNOPSIS

```
#include <curses.h>

int insdelln(int n);
int winsdelln(WINDOW *win, int n);
```

DESCRIPTION
The insdelln() and winsdelln() functions perform the following actions:

- If \( n \) is positive, these functions insert \( n \) lines into the current or specified window before the current line. The \( n \) last lines are no longer displayed.

- If \( n \) is negative, these functions delete \( n \) lines from the current or specified window starting with the current line, and move the remaining lines toward the cursor. The last \( n \) lines are cleared.

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

CHANGE HISTORY
First released in Issue 4.
NAME
insertln, winsertln — insert lines into a window

SYNOPSIS
#include <curses.h>
int insertln(void);
int winsertln(WINDOW *win);

DESCRIPTION
The insertln() and winsertln() 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>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity. The argument list for the insertln() function is explicitly declared as void.
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 insnstr() and winsnstr() functions perform wrapping. The insstr() and winsstr() 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
<curses.h>.

CHANGE HISTORY
First released in Issue 4.
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
<curses.h>.

CHANGE HISTORY
First released in Issue 4.
NAME
insstr — insert a multi-byte character string into the current window

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

DESCRIPTION
Refer to insstr().

CHANGE HISTORY
First released in Issue 4.
NAME
instr — input a multi-byte character string from the current window

SYNOPSIS
#include <curses.h>

int instr(char *str);

DESCRIPTION
Refer to instr().

CHANGE HISTORY
First released in Issue 4.
ins_wch()

NAME
ins_wch, mvins_wch, mvwins_wch, wins_wch — insert a complex character and rendition into a window

SYNOPSIS

```c
#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, 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

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>`.

CHANGE HISTORY

First released in Issue 4.

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()`.

CHANGE HISTORY

First released in Issue 4.

intrflush()  

CURSES  

Curses Interfaces

NAME
intrflush — enable or disable flush on interrupt

SYNOPSIS
#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 the value of bf is TRUE, then flushing of the output buffer associated with the current screen will occur when an interrupt key (interrupt, suspend, or quit) is pressed. If the value of bf is FALSE then no flushing of the buffer will occur when an interrupt key is pressed. 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
Section 3.5 on page 23, <curses.h>, XBD specification, Section 9.2, Parameters That Can Be Set.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity.

Issue 4, Version 2
The description of the bf argument has been changed to align with Issue 3 and preserve compatibility.
NAME

in_wch, mvin_wch, mvwin_wch, win_wch — extract a complex character and rendition from a window

SYNOPSIS

```c
#include <curses.h>

int in_wch(cchar_t *wcval);
int mvin_wch(int y, int x, cchar_t *wcval);
int mvwin_wch(WINDOW *win, int y, int x, cchar_t *wcval);
int win_wch(WINDOW *win, 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>`.

CHANGE HISTORY

First released in Issue 4.
in_wchnstr()  Curses Interfaces

NAME

in_wchnstr, in_wchstr, mvin_wchnstr, mvin_wchstr, mvwin_wchnstr, mvwin_wchstr, win_wchnstr, win_wchstr — extract an array of complex characters and renditions from a window

SYNOPSIS

C

#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>.

CHANGE HISTORY

First released in Issue 4.
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().

CHANGE HISTORY
First released in Issue 4.
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
    endwin(), <curses.h>.

CHANGE HISTORY
    First released in Issue 4.
NAME
is_linetouched, is_wintouched, touchline, touchwin, untouchwin, wtouchln — 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 wtouchln(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 wtouchln(), if changed is 1, touches n lines in the specified window, starting at line y. If changed is 0, wtouchln() 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
Section 3.2 on page 14, doupdate(), <curses.h>.

CHANGE HISTORY
First released in Issue 4.
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>.

CHANGE HISTORY
First released in Issue 4.
NAME
keypad — 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 behaviour 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
Section 3.5.1 on page 23, `<curses.h>`.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity.
NAME

killchar, killwchar — terminal environment query functions

SYNOPSIS

#include <curses.h>

char killchar(void);

int killwchar(wchar_t *ch);

DESCRIPTION

Refer to erasechar().

CHANGE HISTORY

First released in Issue 2.

Issue 4

The functionality previously described by this entry is moved to the entry for erasechar().
NAME
leaveok — control cursor position resulting from refresh operations

SYNOPSIS
#include <curses.h>

int leaveok(WINDOW *win, bool bf);

DESCRIPTION
Refer to clearok().

CHANGE HISTORY
First released in Issue 2.

Issue 4
The functionality previously described by this entry is moved to the entry for clearok().
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>.

CHANGE HISTORY
    First released in Issue 4.
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>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity. The argument list for the longname() function is explicitly declared as void.
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
Section 3.5 on page 23, getch(), <curses.h>, XBD specification, Section 9.2, Parameters That Can
Be Set (ISTRIP flag).

CHANGE HISTORY
First released in Issue 4.
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>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity.
NAME

mv — pointer page for functions with mv prefix

DESCRIPTION

Most cases in which a Curses function has the mv prefix 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>mvwaddch()</td>
</tr>
<tr>
<td>mvaddchnstr()</td>
<td>mvwaddchnstr()</td>
</tr>
<tr>
<td>mvaddchstr()</td>
<td>mvwaddchstr()</td>
</tr>
<tr>
<td>mvaddnstr()</td>
<td>mvwaddnstr()</td>
</tr>
<tr>
<td>mvaddstr()</td>
<td>mvwaddstr()</td>
</tr>
<tr>
<td>mvaddnwstr()</td>
<td>mvwaddnwstr()</td>
</tr>
<tr>
<td>mvaddwstr()</td>
<td>mvwaddwstr()</td>
</tr>
<tr>
<td>mvadd_wch()</td>
<td>mvwadd_wch()</td>
</tr>
<tr>
<td>mvadd_wchnstr()</td>
<td>mvwadd_wchnstr()</td>
</tr>
<tr>
<td>mvadd_wchstr()</td>
<td>mvwadd_wchstr()</td>
</tr>
<tr>
<td>mvchgt()</td>
<td>mvwchgt()</td>
</tr>
<tr>
<td>mvdelch()</td>
<td>mvwdelch()</td>
</tr>
<tr>
<td>mvgetch()</td>
<td>mvwgetch()</td>
</tr>
<tr>
<td>mvgetnstr()</td>
<td>mvwgetnstr()</td>
</tr>
<tr>
<td>mvgetstr()</td>
<td>mvwgetstr()</td>
</tr>
<tr>
<td>mvgetn_wstr()</td>
<td>mvwgetn_wstr()</td>
</tr>
<tr>
<td>mvget_wch()</td>
<td>mvwget_wch()</td>
</tr>
<tr>
<td>mvhline()</td>
<td>mvwhline()</td>
</tr>
<tr>
<td>mvhline_set()</td>
<td>mvwhline_set()</td>
</tr>
<tr>
<td>mvinsch()</td>
<td>mvwinsch()</td>
</tr>
<tr>
<td>mvinsnstr()</td>
<td>mvwinsnstr()</td>
</tr>
<tr>
<td>mvinnstr()</td>
<td>mvwinnstr()</td>
</tr>
<tr>
<td>mvinnwstr()</td>
<td>mvwinnwstr()</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 Interfaces

**mv**

<table>
<thead>
<tr>
<th>Function</th>
<th>Refer to</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvinsstr()</td>
<td>mvwinstr()</td>
</tr>
<tr>
<td>mvins_nwstr()</td>
<td>mvwins_nwstr()</td>
</tr>
<tr>
<td>mvins_wch()</td>
<td>mvwins_wch()</td>
</tr>
<tr>
<td>mvins_wstr()</td>
<td>mvwins_wstr()</td>
</tr>
<tr>
<td>mvins_wchstr()</td>
<td>mvwins_wchstr()</td>
</tr>
<tr>
<td>mvins_wchnstr()</td>
<td>mvwins_wchnstr()</td>
</tr>
<tr>
<td>mvins_wchstr()</td>
<td>mvwins_wchstr()</td>
</tr>
<tr>
<td>mvprintw()</td>
<td>mvwprintw()</td>
</tr>
<tr>
<td>mvscanw()</td>
<td>mvwscanw()</td>
</tr>
<tr>
<td>mvline()</td>
<td>mvwline()</td>
</tr>
<tr>
<td>mvline_set()</td>
<td>mvwline_set()</td>
</tr>
</tbody>
</table>

**SEE ALSO**

w.

**CHANGE HISTORY**

First released in Issue 4.
NAME
mvcur — output cursor movement commands to the terminal

SYNOPSIS
#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>.

CHANGE HISTORY
First released in Issue 4.
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.

RETURN VALUE
Upon successful completion, mvderwin() returns OK. Otherwise, it returns ERR.

ERRORS
No errors are defined.

SEE ALSO
derwin(), doupdate(), dupwin(), <curses.h>.

CHANGE HISTORY
First released in Issue 4.
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() (in the XSH specification), <curses.h>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity and its name is changed from printw() to mvprintw().
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() (in the XSH specification), wcstombs() (in the XSH specification),
<curses.h>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity and its name is changed from scanw() to mvscanw().
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>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity.
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() (in the XSH specification), <curses.h>.

CHANGE HISTORY
First released in Issue 4.
newpad()  

NAME
newpad, pnoutrefresh, prefresh, subpad — pad management functions

SYNOPSIS

```c
#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 specialised 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 `(begin_y, begin_x)` on the pad. The window is made in the middle of the window `orig`, so that changes made to one window affect both windows.

The `prefresh()` and `pnoutrefresh()` functions are analogous to `wrefresh()` and `wnoutrefresh()` except that they relate to pads instead of windows. The additional arguments indicate what part of the pad and screen are involved. The `pminrow` and `pmincol` arguments specify the origin of the rectangle to be displayed in the pad. The `sminrow`, `smincol`, `smaxrow` and `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 `pminrow`, `pmincol`, `sminrow` or `smincol` are treated as if they were zero.

RETURN VALUE
Upon successful completion, the `newpad()` and `subpad()` functions return a pointer to the pad data structure. Otherwise, they return a null pointer.

Upon successful completion, `pnoutrefresh()` and `prefresh()` return OK. Otherwise, they return ERR.

ERRORS
No errors are defined.

APPLICATION USAGE
To refresh a pad, call `prefresh()` or `pnoutrefresh()`, not `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 `touchwin()` or `touchline()` on the pad before calling `prefresh()`.
SEE ALSO

derwin(), doupdate(), is_linetouched(), <curses.h>.

CHANGE HISTORY

First released in Issue 2.

Issue 4

The pnoutrefresh() and refresh() functions are merged with this entry. In previous issues, they appeared in the entry for refresh(). The subpad() function is new in Issue 4.
NAME
	newterm — screen initialisation function

SYNOPSIS

#include <curses.h>

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

DESCRIPTION

Refer to initscr().

CHANGE HISTORY

First released in Issue 2.

Issue 4

The functionality previously described by this entry is moved to the entry for initscr().
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().

CHANGE HISTORY
First released in Issue 2.

Issue 4
The functionality previously described by this entry is moved to the entry for derwin().
NAME
  nl, nonl — enable/disable newline translation

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

DESCRIPTION
  The \textit{nl}() function enables a mode in which carriage return is translated to newline on input. The \textit{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 \textit{nonl}(), the application can sense the pressing of the carriage return key.

SEE ALSO
  <curses.h>.

CHANGE HISTORY
  First released in Issue 2.

  \textbf{Issue 4}
  The entry is rewritten for clarity. The argument list for the \textit{nl}() and \textit{nonl}() functions is explicitly declared as \textbf{void}. 
NAME

no — pointer page for functions with no prefix

DESCRIPTION

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>

CHANGE HISTORY

First released in Issue 4.

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.
nodelay()

CURSES

Curses Interfaces

NAME
nodelay — enable or disable block during read

SYNOPSIS
#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
Section 3.5 on page 23, getch(), halfdelay(), <curses.h>, XBD specification, Section 9.2,
Parameters That Can Be Set.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity.
NAME

noqiflush, qiflush — enable/disable queue flushing

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

Section 3.5 on page 23, `intrflush()`, `<curses.h>`, XBD specification, Section 9.2, Parameters That Can Be Set (NOFLUSH flag).

CHANGE HISTORY

First released in Issue 4.
NAME
notimeout, timeout, wtimeout — control blocking on input

SYNOPSIS
#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
Section 3.5 on page 23, getch(), halfdelay(), nodelay(), <curses.h>, XBD specification, Section 9.2,
Parameters That Can Be Set.

CHANGE HISTORY
First released in Issue 4.
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.

ERRORS
No errors are defined.

SEE ALSO
copywin(), <curses.h>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity. The type of argument srcwin() is changed from WINDOW * to WINDOW *CONST.

NAME
pair_content, PAIR_NUMBER — get information on a colour 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().

CHANGE HISTORY
First released in Issue 4.
NAME
pechochar, pecho_wchar — write a character and rendition and immediately refresh the pad

SYNOPSIS
```c
#include <curses.h>
int pechochar(WINDOW * pad, 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 `prefresh()`. The last location of the pad on the screen is reused for the arguments to `prefresh()`.

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>`.

CHANGE HISTORY
First released in Issue 4.

Issue 4, Version 2
The second argument of `pechochar()` is changed to type `chttype` from `chttype *`. 
pnoutrefresh()  Curses Interfaces

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().

CHANGE HISTORY
First released in Issue 2.

Issue 4
The functionality previously described by this entry is moved to the entry for newpad().
NAME
printw — print formatted output in the current window

SYNOPSIS
#include <curses.h>

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

DESCRIPTION
Refer to mvprintw().

CHANGE HISTORY
First released in Issue 2.

Issue 4
The functionality previously described by this entry is moved to the entry for mvprintw().
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));

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
Changing the terminal attributes using these functions may cause the renditions of characters within a curses window to be altered on some terminals.

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.

SEE ALSO
doupdate(), is_linetouched(), putchar() (in the XSH specification), tgetent(), tigetflag(), <term.h>.

CHANGE HISTORY
First released in Issue 4.

NAME
putwin — dump window to a file

SYNOPSIS

```c
#include <curses.h>

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

DESCRIPTION
Refer to `getwin()`.

CHANGE HISTORY
First released in Issue 4.
NAME
qiflush — enable queue flushing

SYNOPSIS
#include <curses.h>

void qiflush(void);

DESCRIPTION
Refer to noqiflush().

CHANGE HISTORY
First released in Issue 4.
NAME
  raw — set Raw Mode

SYNOPSIS
  #include <curses.h>
  int raw(void);

DESCRIPTION
  Refer to cbreak().

CHANGE HISTORY
  First released in Issue 2.

Issue 4
  The functionality previously described by this entry is moved to the entry for cbreak().
redrawwin() ENHANCED CURSES Curses Interfaces

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 optimisation based on assumptions about what is physically displayed there.

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>`.

CHANGE HISTORY
First released in Issue 4.
NAME
refresh — refresh current window

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

DESCRIPTION
Refer to doupdate().

CHANGE HISTORY
First released in Issue 2.

Issue 4
The functionality previously described by this entry is moved to the entry for doupdate().
reset_prog_mode()  CURSES  Curses Interfaces

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 \textit{def_prog_mode}().

CHANGE HISTORY
First released in Issue 2.

Issue 4
The functionality previously described by this entry is moved to the entry for \textit{def_prog_mode}().
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>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity. The argument list for the resetty() and savetty() functions is explicitly declared as void.
NAME
restartterm — change terminal type

SYNOPSIS
#include <term.h>

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

DESCRIPTION
Refer to del_curterm().

CHANGE HISTORY
First released in Issue 4.
NAME

ripoffline — reserve a line for a dedicated purpose

SYNOPSIS

```c
#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.

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 initialisation 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 initialisation 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_attroff()`, `<curses.h>`.

CHANGE HISTORY

First released in Issue 4.

NAME
savetty — save terminal mode

SYNOPSIS
#include <curses.h>
int savetty(void);

DESCRIPTION
Refer to resetty().

CHANGE HISTORY
First released in Issue 4.
NAME
scanw — convert formatted input from the current window

SYNOPSIS
#include <curses.h>

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

DESCRIPTION
Refer to mvscanw().

CHANGE HISTORY
First released in Issue 2.

Issue 4
The functionality previously described by this entry is moved to the entry for mvscanw().
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 initialise 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 nrmc 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(), douupdate(), endwin(), getwin(), open() (in the XSH specification), read() (in the XSH specification), write() (in the XSH specification), <curses.h>.

CHANGE HISTORY
First released in Issue 4.
NAME

scr1, scroll, wscr1 — scroll a Curses window

SYNOPSIS

```c
#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. The interaction of these functions with `setsccreg()` is currently unspecified.

RETURN VALUE

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

ERRORS

No errors are defined.

SEE ALSO

 `<curses.h>`.

CHANGE HISTORY

First released in Issue 4.

In previous issues, the `scroll()` function was described in an entry of its own. It has been merged with this entry in Issue 4. Its description has been rewritten for clarity, but otherwise its functionality is identical.

FUTURE DIRECTIONS

In a future issue the interaction of these functions with `setsccreg()` will be defined.
scrollok()  CURSES  Curses Interfaces

NAME
   scrollok — enable or disable scrolling on a window

SYNOPSIS
   
   #include <curses.h>

   int scrollok(WINDOW *win, bool bf);

DESCRIPTION
   Refer to clearok().

CHANGE HISTORY
   First released in Issue 2.

Issue 4
   The functionality previously described by this entry is moved to the entry for clearok().
NAME
setcchar — set cchar_t from a wide character string and rendition

SYNOPSIS

```c
#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 initialises the object pointed to by `wcval` according to the character attributes in `attrs`, the colour 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

Section 3.3 on page 16, `attroff()`, `can_change_color()`, `getcchar()`, `<curses.h>`.

CHANGE HISTORY

First released in Issue 4.

NAME
set_curterm — set current terminal

SYNOPSIS
#include <term.h>

TERMINAL *set_curterm(TERMINAL *nterm);

DESCRIPTION
Refer to del_curterm().

CHANGE HISTORY
First released in Issue 4.
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().

CHANGE HISTORY
First released in Issue 2.

Issue 4
The functionality previously described by this entry is moved to the entry for clearok().
NAME
set_term — switch between screens

SYNOPSIS
#include <curses.h>
SCREEN *set_term(SCREEN *new);

DESCRIPTION
The set_term() function switches between different screens. The new argument specifies the new current screen.

RETURN VALUE
Upon successful completion, set_term() returns a pointer to the previous screen. Otherwise, it returns a null pointer.

ERRORS
No errors are defined.

APPLICATION USAGE
This is the only function that manipulates SCREEN pointers; all other functions affect only the current screen.

SEE ALSO
Section 3.2 on page 14, initscr(), <curses.h>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity.
NAME
setupterm — access the terminfo database

SYNOPSIS
#include <term.h>

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

DESCRIPTION
Refer to del_curterm().

CHANGE HISTORY
First released in Issue 4.
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);
int 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);
int 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
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_attrset() and slk_attroff() functions correspond to attron(), attrset(), and attroff(). 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_attrond(), slk_attrset() and color_set() and thus support the attribute constants with the WA_prefix and colour.

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, 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 wcwidth() and 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() (in the XSH specification), ripoffline(), wcswidth() (in the XSH specification), <curses.h>.

CHANGE HISTORY
First released in Issue 4.

Issue 4, Version 2
This entry is rewritten to include the colour handling functions.
NAME
standend, standout, wstandend, wstandout — set and clear window attributes

SYNOPSIS
#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>.

CHANGE HISTORY
Derived from the attroff() entry in Issue 3. The entry is reworded for clarity, but otherwise the
functionality is identical to previous issues.
NAME

start_color — initialise use of colours on terminal

SYNOPSIS

```c
#include <curses.h>

int start_color(void);
```

DESCRIPTION

Refer to `can_change_color()`.

CHANGE HISTORY

First released in Issue 4.
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>.

CHANGE HISTORY
    First released in Issue 4.
NAME
subpad — create a subwindow in a pad

SYNOPSIS

```c
#include <curses.h>

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

DESCRIPTION
Refer to `newpad()`.

CHANGE HISTORY
First released in Issue 4.
NAME
subwin — create a subwindow

SYNOPSIS
#include <curses.h>

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

DESCRIPTION
Refer to derwin().

CHANGE HISTORY
First released in Issue 2.

Issue 4
The functionality previously described by this entry is moved to the entry for derwin().
NAME
  syncok, wcursyncup, wsyncdown, wsncup — synchronise a window with its parents or children

SYNOPSIS
#include <curses.h>

int syncok(WINDOW *win, bool bf);
void wcursyncup(WINDOW *win);
void wsyncdown(WINDOW *win);
void wsncup(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 bf is TRUE, such implicit touching occurs. If 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 win to reflect the current cursor position of win.

  The wsyncdown() function touches win if any ancestor window has been touched.

  The wsncup() function unconditionally touches all ancestors of 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>.

CHANGE HISTORY
  First released in Issue 4.
NAME

termattrs — get supported terminal video attributes

SYNOPSIS

```c
#include <curses.h>

cttype termattrs(void);
attr_t term_attrs(void);
```

DESCRIPTION

The `termattrs()` function extracts the video attributes of the current terminal which is supported by the `cttype` data type.

The `term_attrs()` function extracts information for the video attributes of the current terminal which is supported for a `cchar_t`.

RETURN VALUE

The `termattrs()` function returns a logical OR of A_ values of all video attributes supported by the terminal.

The `term_attrs()` function returns a logical OR of WA_ values of all video attributes supported by the terminal.

ERRORS

No errors are defined.

SEE ALSO

Attributes in `attroff()`, `attr_get()` and `<curses.h>`

CHANGE HISTORY

First released in Issue 4.

NAME
termname — get terminal name

SYNOPSIS

```
#include <curses.h>

char *termname(void);
```

DESCRIPTION

The termname() function obtains the terminal name as recorded by setupterm().

RETURN VALUE

The termname() function returns a pointer to the terminal name.

ERRORS

No errors are defined.

SEE ALSO

Section 6.1.1 on page 239, del_curterm(), getenv() (in the XSH specification), initscr(), <curses.h>.

CHANGE HISTORY

First released in Issue 4.
tgetent( )

ENHANCED CURSES

Curses Interfaces

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 the 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 tigetflg() page.

Portable applications must use tputs() to output the strings returned by tgetstr() and tgoto().

SEE ALSO

putc(), setupterm(), tigetflg(), <term.h>

CHANGE HISTORY

First released in Issue 4.

X/Open CAE Specification 1996
Issue 4, Version 2

The first argument to `tgoto()` is changed from `const char *` to `char *`. 
NAME
tigetflag, tigetnum, tigetstr, tparm — retrieve capabilities from the terminfo database

SYNOPSIS

```c
#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 in the table in Section 6.1.3 on page 241.

The tparm() function takes as cap a string capability. If cap is parameterised (as described in Section A.1.2 on page 256), tparm() resolves the parameterisation. If the parameterised 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 parameterisation resolved. Otherwise, it returns a null pointer.

ERRORS

No errors are defined.

APPLICATION USAGE

For parameterised 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 parameterised 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>.

CHANGE HISTORY

First released in Issue 4.
NAME
  timeout — control blocking on input

SYNOPSIS
  #include <curses.h>
  void timeout(int delay);

DESCRIPTION
  Refer to notimeout().

CHANGE HISTORY
  First released in Issue 4.
NAME
touchline, touchwin — window refresh control functions

SYNOPSIS
#include <curses.h>

int touchline(WINDOW * win, int start, int count);
int touchwin(WINDOW * win);

DESCRIPTION
Refer to is_linetouched().

CHANGE HISTORY
First released in Issue 4.
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().

CHANGE HISTORY
First released in Issue 4.
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().

CHANGE HISTORY
First released in Issue 4.
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
Section 3.5 on page 23, doupdate(), getch(), initscr(), <curses.h>, XBD specification, Section 9.2, Parameters That Can Be Set.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity. The RETURN VALUE section now states that the function returns OK on success and ERR on failure. No return values were defined in previous issues.
NAME
unctrl — generate printable representation of a character

SYNOPSIS
#include <unctrl.h>
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>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity. The RETURN VALUE section now states that the function may return a null pointer. This condition was not specified in previous issues.
ungetch() ENHANCED CURSES Curses Interfaces

NAME  
ungetch, unget_wch — push a character onto the input queue

SYNOPSIS  

```c
#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.

The `unget_wch()` function pushes the wide character `wch` onto the head of the input queue.

One character of push-back is guaranteed. The result of successive calls without an intervening call to `getch()` or `get_wch()` are unspecified.

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

ERRORS  
No errors are defined.

SEE ALSO  
Section 3.5 on page 23, `getch()`, `get_wch()`, `<curses.h>`.

CHANGE HISTORY  
First released in Issue 4.
NAME
untouchwin — window refresh control function

SYNOPSIS
#include <curses.h>

int untouchwin(WINDOW *win);

DESCRIPTION
Refer to is_linetouched().

CHANGE HISTORY
First released in Issue 4.
NAME

use_env — specify source of screen size information

SYNOPSIS

```c
#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>`.

CHANGE HISTORY

First released in Issue 4.

Issue 4, Version 2

The first argument is changed from `char bool` to `bool boolval`. 
NAME
vidattr, vid_attr, vidputs, vid_puts — output attributes to the terminal

SYNOPSIS

```c
#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 color_pair_number, void *opt, int (*putfunc)(int));
```

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 which 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 colour 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` (which can be specified as an argument to either `vidputs()` or `vid_puts()`) is either `putchar()` or some other function with the same prototype. Both the `vidputs()` and the `vid_puts()` function ignore 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() (in the XSH specification), putwchar() (in the XSH specification), tigetflag(), <curses.h>.
CHANGE HISTORY

First released in Issue 4.

Issue 4, Version 2

This entry is rewritten to include the colour handling functions.
NAME
vline — draw vertical line

SYNOPSIS
#include <curses.h>
int vline(chtype ch, int n);

DESCRIPTION
Refer to hline().

CHANGE HISTORY
First released in Issue 4.
NAME
vline_set — draw vertical line from complex character and rendition

SYNOPSIS

```c
#include <curses.h>

int vline_set(const cchar_t *ch, int n);
```

DESCRIPTION
Refer to `hline_set()`.

CHANGE HISTORY
First released in Issue 4.
NAME
vwprintw — print formatted output in window (TO BE WITHDRAWN)

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 requirement to include varargs.h and stdarg.h which both contain definitions of va_list.

SEE ALSO
mvprintw(), fprintf() (in the XSH specification), vw_printw(), <curses.h>, <varargs.h> (in the XSH specification).

CHANGE HISTORY
First released in Issue 4.
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 wprintw(). The use of the wprintw() 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() (in the XSH specification), <curses.h>, <stdarg.h> (in the XSH specification).

CHANGE HISTORY
First released in Issue 4.
NAME
vwscanw — convert formatted input from a window (TO BE WITHDRAWN)

SYNOPSIS

```c
#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()` (in the XSH specification), `mvscanw()`, `vw_scanw()`, `<curses.h>`, `<varargs.h>` (in the XSH specification).

CHANGE HISTORY
First released in Issue 4.
NAME
vw_scanw — convert formatted input from a window

SYNOPSIS

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

int vw_scanw(WINDOW *, char *, va_list varflist);
```

DESCRIPTION

The `vw_scanw` function achieves the same effect as `wscanf()` 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 `vwscanf()`. The use of the `vwscanf()` 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()` (in the XSH specification), `mvscanf()`, `<curses.h>`, `<stdarg.h>` (in the XSH specification).

CHANGE HISTORY

First released in Issue 4.

Corrections made to sections "SYNOPSIS" and "APPLICATION USAGE", Issue 4, Version 2.
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>waddch()</td>
<td>addch()</td>
</tr>
<tr>
<td>waddchnstr()</td>
<td>addchnstr()</td>
</tr>
<tr>
<td>waddchstr()</td>
<td>addchstr()</td>
</tr>
<tr>
<td>waddnstr()</td>
<td>addnstr()</td>
</tr>
<tr>
<td>waddstr()</td>
<td>addnstr()</td>
</tr>
<tr>
<td>waddwstr()</td>
<td>addnstr()</td>
</tr>
<tr>
<td>wadd_wch()</td>
<td>add_wch()</td>
</tr>
<tr>
<td>wadd_wchnstr()</td>
<td>add_wchnstr()</td>
</tr>
<tr>
<td>wadd_wchstr()</td>
<td>add_wchstr()</td>
</tr>
<tr>
<td>wattroff()</td>
<td>attroff()</td>
</tr>
<tr>
<td>wattron()</td>
<td>attron()</td>
</tr>
<tr>
<td>wattrset()</td>
<td>attrset()</td>
</tr>
<tr>
<td>wattr_get()</td>
<td>attr_get()</td>
</tr>
<tr>
<td>wattr_off()</td>
<td>attr_off()</td>
</tr>
<tr>
<td>wattr_on()</td>
<td>attr_on()</td>
</tr>
<tr>
<td>wattr_set()</td>
<td>attr_set()</td>
</tr>
<tr>
<td>wbkgd()</td>
<td>bkgd()</td>
</tr>
<tr>
<td>wbkgdset()</td>
<td>bkgdset()</td>
</tr>
<tr>
<td>wbkgrnd()</td>
<td>bkgrnd()</td>
</tr>
<tr>
<td>wbkgrndset()</td>
<td>bkgrndset()</td>
</tr>
<tr>
<td>wborder()</td>
<td>border()</td>
</tr>
<tr>
<td>wborder_set()</td>
<td>border_set()</td>
</tr>
<tr>
<td>wchcat()</td>
<td>chgat()</td>
</tr>
<tr>
<td>wclear()</td>
<td>clear()</td>
</tr>
<tr>
<td>wcrltobot()</td>
<td>clrtobot()</td>
</tr>
<tr>
<td>wcrltowol()</td>
<td>clrtowol()</td>
</tr>
<tr>
<td>wcursyncup() *</td>
<td>syncok()</td>
</tr>
<tr>
<td>wdelch()</td>
<td>delch()</td>
</tr>
<tr>
<td>wdeleteln()</td>
<td>deleteln()</td>
</tr>
<tr>
<td>wechochar()</td>
<td>echochar()</td>
</tr>
<tr>
<td>wecho_wchar()</td>
<td>echo_wchar()</td>
</tr>
</tbody>
</table>

3. The wunctrl() function is an exception to this rule and has an entry under its own name.

* There is no corresponding function without the w prefix.
<table>
<thead>
<tr>
<th>Function</th>
<th>Refer to</th>
</tr>
</thead>
<tbody>
<tr>
<td>werase()</td>
<td>clear()</td>
</tr>
<tr>
<td>wgetbkgrnd()</td>
<td>bgkgrnd()</td>
</tr>
<tr>
<td>wgetch()</td>
<td>getch()</td>
</tr>
<tr>
<td>wgetnstr()</td>
<td>getnstr()</td>
</tr>
<tr>
<td>wgetn_wstr()</td>
<td>getn_wstr()</td>
</tr>
<tr>
<td>wgetstr()</td>
<td>getnstr()</td>
</tr>
<tr>
<td>wget_wch()</td>
<td>get_wch()</td>
</tr>
<tr>
<td>wget_wstr()</td>
<td>getn_wstr()</td>
</tr>
<tr>
<td>whline()</td>
<td>hline()</td>
</tr>
<tr>
<td>whline_set()</td>
<td>hline_set()</td>
</tr>
<tr>
<td>winch()</td>
<td>inch()</td>
</tr>
<tr>
<td>winchnstr()</td>
<td>inchnstr()</td>
</tr>
<tr>
<td>winchstr()</td>
<td>inchnstr()</td>
</tr>
<tr>
<td>winnstr()</td>
<td>innstr()</td>
</tr>
<tr>
<td>winnwstr()</td>
<td>innwstr()</td>
</tr>
<tr>
<td>winsch()</td>
<td>insch()</td>
</tr>
<tr>
<td>winsdelln()</td>
<td>insdelln()</td>
</tr>
<tr>
<td>winsertln()</td>
<td>insertln()</td>
</tr>
<tr>
<td>winsnstr()</td>
<td>insnstr()</td>
</tr>
<tr>
<td>winstr()</td>
<td>innstr()</td>
</tr>
<tr>
<td>wins_nwstr()</td>
<td>ins_nwstr()</td>
</tr>
<tr>
<td>wins_wch()</td>
<td>ins_wch()</td>
</tr>
<tr>
<td>wins_wstr()</td>
<td>ins_nwstr()</td>
</tr>
<tr>
<td>winwstr()</td>
<td>innwstr()</td>
</tr>
<tr>
<td>win_wch()</td>
<td>in_wch()</td>
</tr>
<tr>
<td>win_wchnstr()</td>
<td>in_wchnstr()</td>
</tr>
<tr>
<td>win_wchstr()</td>
<td>in_wchnstr()</td>
</tr>
<tr>
<td>wmove()</td>
<td>move()</td>
</tr>
<tr>
<td>wnoutrefresh() *</td>
<td>doupdate()</td>
</tr>
<tr>
<td>wprintw()</td>
<td>mvprintw()</td>
</tr>
<tr>
<td>wredrawln()</td>
<td>redrawln()</td>
</tr>
<tr>
<td>wrefresh()</td>
<td>doupdate()</td>
</tr>
<tr>
<td>wscanw()</td>
<td>mscanw()</td>
</tr>
<tr>
<td>wscr()</td>
<td>scrl()</td>
</tr>
<tr>
<td>wsetscreg()</td>
<td>clearok()</td>
</tr>
<tr>
<td>wstandend()</td>
<td>standend()</td>
</tr>
<tr>
<td>wstandout()</td>
<td>standend()</td>
</tr>
<tr>
<td>wsyncdown() *</td>
<td>syncok()</td>
</tr>
<tr>
<td>wsyncup() *</td>
<td>syncok()</td>
</tr>
<tr>
<td>wtimeout()</td>
<td>notimeout()</td>
</tr>
<tr>
<td>wtouchln() *</td>
<td>is_linetouch()</td>
</tr>
<tr>
<td>wvline()</td>
<td>hline()</td>
</tr>
<tr>
<td>wvline_set()</td>
<td>hline_set()</td>
</tr>
</tbody>
</table>

* There is no corresponding function without the w prefix.
CHANGE HISTORY

First released in Issue 4.
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>`.

CHANGE HISTORY

First released in Issue 4.
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 4 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.
NAME
curses.h — definitions for screen handling and optimisation functions

SYNOPSIS
#include <curses.h>

DESCRIPTION

Objects
The <curses.h> header provides a declaration for COLOR_PAIRS, COLORS, COLS, cursor, 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:
(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 colour-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

These data types are described in more detail in Section 2.4 on page 12.

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`:

<table>
<thead>
<tr>
<th>Symbolic Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA_ALTCHARSET</td>
<td>Alternate character set</td>
</tr>
<tr>
<td>WA_BLINK</td>
<td>Blinking</td>
</tr>
<tr>
<td>WA_BOLD</td>
<td>Extra bright or bold</td>
</tr>
<tr>
<td>WA_DIM</td>
<td>Half bright</td>
</tr>
<tr>
<td>WA_HORIZONTAL</td>
<td>Horizontal highlight</td>
</tr>
<tr>
<td>WA_INVIS</td>
<td>Invisible</td>
</tr>
<tr>
<td>WA_LEFT</td>
<td>Left highlight</td>
</tr>
<tr>
<td>WA_LOW</td>
<td>Low highlight</td>
</tr>
<tr>
<td>WA_PROTECT</td>
<td>Protected</td>
</tr>
<tr>
<td>WA_REVERSE</td>
<td>Reverse video</td>
</tr>
<tr>
<td>WA_RIGHT</td>
<td>Right highlight</td>
</tr>
<tr>
<td>WA_STANDOUT</td>
<td>Best highlighting mode of the terminal</td>
</tr>
<tr>
<td>WA_TOP</td>
<td>Top highlight</td>
</tr>
<tr>
<td>WA_UNDERLINE</td>
<td>Underlining</td>
</tr>
<tr>
<td>WA_VERTICAL</td>
<td>Vertical highlight</td>
</tr>
</tbody>
</table>

These attribute flags shall be distinct.

The following symbolic constants are used to manipulate attribute bits in objects of type `chtype`:

<table>
<thead>
<tr>
<th>Symbolic Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_ALTCHARSET</td>
<td>Alternate character set</td>
</tr>
<tr>
<td>A_BLINK</td>
<td>Blinking</td>
</tr>
<tr>
<td>A_BOLD</td>
<td>Extra bright or bold</td>
</tr>
<tr>
<td>A_DIM</td>
<td>Half bright</td>
</tr>
<tr>
<td>A_INVIS</td>
<td>Invisible</td>
</tr>
<tr>
<td>A_PROTECT</td>
<td>Protected</td>
</tr>
<tr>
<td>A_REVERSE</td>
<td>Reverse video</td>
</tr>
<tr>
<td>A_STANDOUT</td>
<td>Best highlighting mode of the terminal</td>
</tr>
<tr>
<td>A_UNDERLINE</td>
<td>Underlining</td>
</tr>
</tbody>
</table>

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 `chtype`:

<table>
<thead>
<tr>
<th>Symbolic Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_ATTRIBUTES</td>
<td>Bit-mask to extract attributes</td>
</tr>
<tr>
<td>A_CHARGETXT</td>
<td>Bit-mask to extract a character</td>
</tr>
<tr>
<td>A_COLOR</td>
<td>Bit-mask to extract colour-pair information</td>
</tr>
</tbody>
</table>
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 as described in Section A.1.12 on page 264.

<table>
<thead>
<tr>
<th>char Constant</th>
<th>cchar_t Constant</th>
<th>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>+</td>
<td>plus</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>
Colour-related Macros

The following colour-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 $0 \leq n \leq 63$</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>
<tr>
<td>KEY_PPAGE</td>
<td>Previous page</td>
</tr>
<tr>
<td>KEY_TAB</td>
<td>Set tab</td>
</tr>
<tr>
<td>KEY_CTAB</td>
<td>Clear tab</td>
</tr>
<tr>
<td>KEY_CATAB</td>
<td>Clear all tabs</td>
</tr>
<tr>
<td>KEY_ENTER</td>
<td>Enter or send</td>
</tr>
<tr>
<td>KEY_SRESET</td>
<td>Soft (partial) reset</td>
</tr>
<tr>
<td>KEY_RESET</td>
<td>Reset or hard reset</td>
</tr>
<tr>
<td>KEY_PRINT</td>
<td>Print or copy</td>
</tr>
<tr>
<td>KEY_LL</td>
<td>Home down or bottom</td>
</tr>
<tr>
<td>KEY_A1</td>
<td>Upper left of keypad</td>
</tr>
<tr>
<td>KEY_A3</td>
<td>Upper right of keypad</td>
</tr>
<tr>
<td>KEY_B2</td>
<td>Center of keypad</td>
</tr>
<tr>
<td>KEY_C1</td>
<td>Lower left of keypad</td>
</tr>
<tr>
<td>KEY_C2</td>
<td>Lower right of keypad</td>
</tr>
</tbody>
</table>

The virtual keypad is a 3-by-3 keypad arranged as follows:

```
  A1  UP  A3
  LEFT B2 RIGHT
    C1  DOWN  C3
```

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>
<tr>
<td>KEY_COMMAND</td>
<td>Cmd (command) key</td>
</tr>
<tr>
<td>KEY_COPY</td>
<td>Copy key</td>
</tr>
<tr>
<td>KEY_CREATE</td>
<td>Create key</td>
</tr>
<tr>
<td>KEY_END</td>
<td>End key</td>
</tr>
<tr>
<td>KEY_EXIT</td>
<td>Exit key</td>
</tr>
<tr>
<td>KEY_FIND</td>
<td>Find key</td>
</tr>
<tr>
<td>KEY_HELP</td>
<td>Help key</td>
</tr>
<tr>
<td>KEY_MARK</td>
<td>Mark key</td>
</tr>
<tr>
<td>KEY_MESSAGE</td>
<td>Message key</td>
</tr>
<tr>
<td>KEY_MOVE</td>
<td>Move key</td>
</tr>
<tr>
<td>KEY_NEXT</td>
<td>Next object key</td>
</tr>
<tr>
<td>KEY_OPEN</td>
<td>Open key</td>
</tr>
<tr>
<td>KEY_OPTIONS</td>
<td>Options key</td>
</tr>
<tr>
<td>KEY_PREVIOUS</td>
<td>Previous object key</td>
</tr>
<tr>
<td>KEY_REDO</td>
<td>Redo key</td>
</tr>
<tr>
<td>KEY_REFERENCE</td>
<td>Reference key</td>
</tr>
<tr>
<td>KEY_REFRESH</td>
<td>Refresh key</td>
</tr>
<tr>
<td>KEY_REPLACE</td>
<td>Replace key</td>
</tr>
<tr>
<td>KEY_RESTART</td>
<td>Restart key</td>
</tr>
<tr>
<td>KEY_RESUME</td>
<td>Resume key</td>
</tr>
<tr>
<td>KEY_SAVE</td>
<td>Save key</td>
</tr>
<tr>
<td>KEY_SBEG</td>
<td>Shifted beginning key</td>
</tr>
<tr>
<td>KEY_SCANCEL</td>
<td>Shifted cancel key</td>
</tr>
<tr>
<td>KEY_SCOMMAND</td>
<td>Shifted command key</td>
</tr>
<tr>
<td>KEY_SCOPY</td>
<td>Shifted copy key</td>
</tr>
<tr>
<td>KEY_SCREATE</td>
<td>Shifted create key</td>
</tr>
<tr>
<td>KEY_SDC</td>
<td>Shifted delete char key</td>
</tr>
<tr>
<td>KEY(SDL)</td>
<td>Shifted delete line key</td>
</tr>
<tr>
<td>KEY_SELECT</td>
<td>Select key</td>
</tr>
<tr>
<td>KEY_SEND</td>
<td>Shifted end key</td>
</tr>
<tr>
<td>KEY_SEOL</td>
<td>Shifted clear line key</td>
</tr>
<tr>
<td>KEY_SEXIT</td>
<td>Shifted exit key</td>
</tr>
<tr>
<td>KEY_SFIND</td>
<td>Shifted find key</td>
</tr>
<tr>
<td>KEY_SHELP</td>
<td>Shifted help key</td>
</tr>
<tr>
<td>KEY_SHOME</td>
<td>Shifted home key</td>
</tr>
<tr>
<td>KEY_SIC</td>
<td>Shifted input key</td>
</tr>
<tr>
<td>KEY_SLEFT</td>
<td>Shifted left arrow key</td>
</tr>
<tr>
<td>KEY_SMESSAGE</td>
<td>Shifted message key</td>
</tr>
<tr>
<td>KEY_SMOVE</td>
<td>Shifted move key</td>
</tr>
<tr>
<td>KEY_SNEXT</td>
<td>Shifted next key</td>
</tr>
<tr>
<td>KEY_SOPTIONS</td>
<td>Shifted options key</td>
</tr>
<tr>
<td>Key Code</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>KEY_SPREVIOUS</td>
<td>Shifted prev key</td>
</tr>
<tr>
<td>KEY_SPRINT</td>
<td>Shifted print key</td>
</tr>
<tr>
<td>KEY_SREDO</td>
<td>Shifted redo key</td>
</tr>
<tr>
<td>KEY_SREPLACE</td>
<td>Shifted replace key</td>
</tr>
<tr>
<td>KEY_SRIGHT</td>
<td>Shifted right arrow</td>
</tr>
<tr>
<td>KEY_SRSUME</td>
<td>Shifted resume key</td>
</tr>
<tr>
<td>KEY_SSAVE</td>
<td>Shifted save key</td>
</tr>
<tr>
<td>KEY_SSUSPEND</td>
<td>Shifted suspend key</td>
</tr>
<tr>
<td>KEY_SUNDO</td>
<td>Shifted undo key</td>
</tr>
<tr>
<td>KEY_SUSPEND</td>
<td>Suspend key</td>
</tr>
<tr>
<td>KEY_UNDO</td>
<td>Undo key</td>
</tr>
</tbody>
</table>
Function Prototypes

The following are declared as functions, and may also be defined as macros:

```c
int addch(const chtype);
int addchnstr(const chtype *, int);
int addchstr(const chtype *);
int addchnstr(const chtype *, int);
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);
int bkgrnd(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 *);
b Anglo
```
Curses.h

EC

int delay_output(int);
int delech(void);
int deleteln(void);

void delscreen(SCREEN *);
int delwin(WINDOW *);

WINDOW *derwin(WINDOW *, 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);
char erasechar(void);
int erase(void);

int erasewchar(wchar_t *);
void filter(void);
int flash(void);
int flushinp(void);

chttype getbkgd(WINDOW *);
int getbkgrnd(cchar_t *);
int getcchar(const cchar_t *, wchar_t *, attr_t *, short *, void *);
int getch(void);

int getnstr(char *, int);
int getn_wstr(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 *, bool);

void immedok(WINDOW *, bool);

chttype 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);

chttype insch(chtype);

int insdelln(int);
int insertln(void);

int insnstr(const char *, int);
int insnwstr(const wchar_t *, int);

int insstr(const char *);
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int ins_wch(const cchar_t *)</code></td>
<td>Insert character without echoes</td>
</tr>
<tr>
<td><code>int ins_wstr(const wchar_t *)</code></td>
<td>Insert string without echoes</td>
</tr>
<tr>
<td><code>int intrflush(WINDOW *, bool)</code></td>
<td>Insert output flush</td>
</tr>
<tr>
<td><code>int in_wch(cchar_t *)</code></td>
<td>Insert character</td>
</tr>
<tr>
<td><code>int in_wstr(const wchar_t *)</code></td>
<td>Insert string</td>
</tr>
<tr>
<td><code>int intrflush(WINDOW *, bool)</code></td>
<td>Insert output flush</td>
</tr>
<tr>
<td><code>int in_wchnstr(cchar_t *, int)</code></td>
<td>Insert character with newline</td>
</tr>
<tr>
<td><code>int in_wchstr(cchar_t *)</code></td>
<td>Insert character string</td>
</tr>
<tr>
<td><code>int inwstr(wchar_t *)</code></td>
<td>Insert string</td>
</tr>
<tr>
<td><code>bool isendwin(void)</code></td>
<td>Check if window is ended</td>
</tr>
<tr>
<td><code>bool is_linetouched(WINDOW *, int)</code></td>
<td>Check if line is touched</td>
</tr>
<tr>
<td><code>bool is_wintouched(WINDOW *)</code></td>
<td>Check if window is touched</td>
</tr>
<tr>
<td><code>char *keyname(int)</code></td>
<td>Get key name</td>
</tr>
<tr>
<td><code>char *key_name(wchar_t)</code></td>
<td>Get key name</td>
</tr>
<tr>
<td><code>int keypad(WINDOW *, bool)</code></td>
<td>Enable keypad</td>
</tr>
<tr>
<td><code>int killchar(void)</code></td>
<td>Kill character</td>
</tr>
<tr>
<td><code>int killwchar(wchar_t *)</code></td>
<td>Kill wide character</td>
</tr>
<tr>
<td><code>int leaveok(WINDOW *, bool)</code></td>
<td>Leave OK</td>
</tr>
<tr>
<td><code>char *longname(void)</code></td>
<td>Get long name</td>
</tr>
<tr>
<td><code>int meta(WINDOW *, bool)</code></td>
<td>Enable meta</td>
</tr>
<tr>
<td><code>int move(int, int)</code></td>
<td>Move window</td>
</tr>
<tr>
<td><code>int mvaddch(int, int, const chtype)</code></td>
<td>Move character</td>
</tr>
<tr>
<td><code>int mvaddchnstr(int, int, const chtype *, int)</code></td>
<td>Move character with newline</td>
</tr>
<tr>
<td><code>int mvaddchstr(int, int, const chtype *)</code></td>
<td>Move character string</td>
</tr>
<tr>
<td><code>int mvaddnstr(int, int, const char *, int)</code></td>
<td>Move character</td>
</tr>
<tr>
<td><code>int mvaddnstrstr(int, int, const wchar_t *, int)</code></td>
<td>Move character string</td>
</tr>
<tr>
<td><code>int mvaddwch(int, int, const cchar_t *)</code></td>
<td>Move wide character</td>
</tr>
<tr>
<td><code>int mvaddweekday(int, int, const cchar_t *, int)</code></td>
<td>Move wide character with newline</td>
</tr>
<tr>
<td><code>int mvaddwchstr(int, int, const cchar_t *)</code></td>
<td>Move wide character string</td>
</tr>
<tr>
<td><code>int mvaddwstr(int, int, const wchar_t *)</code></td>
<td>Move wide string</td>
</tr>
<tr>
<td><code>int mvchgat(int, int, int, attr_t, short, const void *)</code></td>
<td>Move character with attributes</td>
</tr>
<tr>
<td><code>int mvcur(int, int, int, int)</code></td>
<td>Move cursor</td>
</tr>
<tr>
<td><code>int mvdelch(int, int)</code></td>
<td>Delete character</td>
</tr>
<tr>
<td><code>int mvderwin(WINDOW *, int, int)</code></td>
<td>Derive window</td>
</tr>
<tr>
<td><code>int mvinch(int, int)</code></td>
<td>Move input</td>
</tr>
<tr>
<td><code>int mvgetnstr(int, int, char *, int)</code></td>
<td>Get next string</td>
</tr>
<tr>
<td><code>int mvgetnstrstr(int, int, null) </code></td>
<td>Get next string</td>
</tr>
<tr>
<td><code>int mvinstr(int, int, char *, int)</code></td>
<td>Get input</td>
</tr>
<tr>
<td><code>int mvinstr(int, int, const char *, int)</code></td>
<td>Get input with attributes</td>
</tr>
<tr>
<td><code>int mvinsch(int, int, chtype)</code></td>
<td>Insert character</td>
</tr>
<tr>
<td><code>int mvinsnstr(int, int, const char *, int)</code></td>
<td>Insert character with newline</td>
</tr>
<tr>
<td><code>int mvinsnstrstr(int, int, const wchar_t *, int)</code></td>
<td>Insert character string</td>
</tr>
<tr>
<td><code>int mvinsstr(int, int, const char *, int)</code></td>
<td>Insert string</td>
</tr>
<tr>
<td><code>int mvinsstr(int, int, const cchar_t *)</code></td>
<td>Insert string</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td><code>int mvins_wstr(int, int, const wchar_t *);</code></td>
<td>Insert a wide string into the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvin_wch(int, int, cchar_t *);</code></td>
<td>Insert a character into the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvin_wchnstr(int, int, cchar_t *, int);</code></td>
<td>Insert a wide string into the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvin_wchstr(int, int, cchar_t *);</code></td>
<td>Insert a wide string into the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvinwstr(int, int, wchar_t *);</code></td>
<td>Insert a wide string into the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvprintw(int, int, char *, ...);</code></td>
<td>Print a wide string into the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvscanw(int, int, char *, ...);</code></td>
<td>Scan a wide string from the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvvline(int, int, ctype, int);</code></td>
<td>Move the cursor to the specified line and column</td>
</tr>
<tr>
<td><code>int mvvline_set(int, int, const cchar_t *, int);</code></td>
<td>Set the cursor position to the specified coordinates</td>
</tr>
<tr>
<td><code>int mvwaddch(WINDOW *, int, int, const ctype);</code></td>
<td>Add a character to the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvwaddchnstr(WINDOW *, int, int, const cchar_t *, int);</code></td>
<td>Add a wide string to the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvwaddchstr(WINDOW *, int, int, const chtype *);</code></td>
<td>Add a character to the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvwaddnstr(WINDOW *, int, int, const char *);</code></td>
<td>Add a string to the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvwaddnwstr(WINDOW *, int, int, const wchar_t *);</code></td>
<td>Add a wide string to the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvwaddstr(WINDOW *, int, int, const char *);</code></td>
<td>Add a string to the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvwadd_wch(WINDOW *, int, int, const cchar_t *);</code></td>
<td>Add a character to the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvwadd_wchnstr(WINDOW *, int, int, const cchar_t *, int);</code></td>
<td>Add a wide string to the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvwadd_wchstr(WINDOW *, int, int, const cchar_t *);</code></td>
<td>Add a character to the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvwaddwstr(WINDOW *, int, int, const wchar_t *);</code></td>
<td>Add a wide string to the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvwchgat(WINDOW *, int, int, attr_t, short, const void *);</code></td>
<td>Change attribute at specified coordinates</td>
</tr>
<tr>
<td><code>int mvwdelch(WINDOW *, int, int);</code></td>
<td>Delete a character at specified coordinates</td>
</tr>
<tr>
<td><code>int mvwgetch(WINDOW *, int, int);</code></td>
<td>Get a character from the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvwgetnstr(WINDOW *, int, int, char *, int);</code></td>
<td>Get a string from the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvwgetn_wstr(WINDOW *, int, int, wint_t *, int);</code></td>
<td>Get a wide string from the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvwgetstr(WINDOW *, int, int, char *);</code></td>
<td>Get a string from the window at specified coordinates</td>
</tr>
<tr>
<td><code>int mvwvline(WINDOW *, int, int, ctype, int);</code></td>
<td>Move the cursor to the specified line and column</td>
</tr>
<tr>
<td><code>int mvwvline_set(WINDOW *, int, int, const cchar_t *, int);</code></td>
<td>Set the cursor position to the specified coordinates</td>
</tr>
<tr>
<td><code>int napms(int);</code></td>
<td>Wait for specified number of milliseconds</td>
</tr>
<tr>
<td><code>WINDOW *newpad(int, int);</code></td>
<td>Create a new pad window with specified dimensions</td>
</tr>
</tbody>
</table>
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 noqiflush(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 *, ctype);
int pecho_wchar(WINDOW *, const cchar_t *);
int pintw(char *, ...);

int putp(const char *);
int putwin(WINDOW *, FILE *);
void qiflush(void);
int raw(void);

int redrawwin(WINDOW *);
int refresh(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 scrl(int);
int scroll(WINDOW *);
int scrollok(WINDOW *, bool);

int scr_restore(const char *);
int scr_set(const char *);
int setcchar(cchar_t *, const wchar_t *, const attr_t, short, const void *);

int setscrreg(int, int);
SCREEN *set_term(SCREEN *);
int setupterm(char *, int, int *);

int slk_attr_off(const attr_t, void *);
int slk_attoff(const chtype);
int slk_attr_on(const attr_t, void *);
int slk_attron(const chtype);
int slk_attr_set(const attr_t, short, void *);
int slk_atrset(const chtype);
int slk_clear(void);
int slk_color(short);
int slk_init(int);
char *slk_label(int);
int slk_noutrefresh(void);
int slk_refresh(void);
int slk_restore(void);
int slk_set(int, const char *, int);
int slk_touch(void);
int slk_wset(int, const wchar_t *, int);
int standend(void);
int standout(void);

int start_color(void);
WINDOW *subpad(WINDOW *, int, int, int, int);
WINDOW *subwin(WINDOW *, int, int, int, int);

int syncok(WINDOW *, bool);
ctime termattrs(void);
attr_t term_attrs(void);
char *termname(void);
tigetflag(char *);
tigetnum(char *);
tigetstr(char *);
void timeout(int);
int touchline(WINDOW *, int, int);
int touchwin(WINDOW *);

char *tparm(char *, long, long, long, long, long, long, long, long);

int ungetch(int);
int unget_wch(const wchar_t);
int untouchwin(WINDOW *);
void use_env(bool);
int vid_attr(attr_t, short, void *);
int vidattr(ctype);
int vid_puts(attr_t, short, void *, int (*)(int));
int vidputs(ctype, int (*)(int));
int vline(ctype, int);

int vline_set(const cchar_t *, int);
int vwprintw(WINDOW *, char *, va_list *);
int vw_printw(WINDOW *, char *, va_list *);
int vwscanw(WINDOW *, char *, va_list *);
int vw_scanw(WINDOW *, char *, va_list *);
int waddch(WINDOW *, const chtype);

int waddchnstr(WINDOW *, const chtype *, int);
int waddchstr(WINDOW *, const chtype *);

int waddnstr(WINDOW *, const char *, int);
int waddnwstr(WINDOW *, const wchar_t *, int);
int waddstr(WINDOW *, const char *);
int wadd_wch(WINDOW *, const cchar_t *);
int wadd_wchnstr(WINDOW *, const cchar_t *, int);
int wadd_wchstr(WINDOW *, const cchar_t *);
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 wbkgrnd(WINDOW *, const cchar_t *);
void wbkgrndset(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 *, const cchar_t *);
int wchgt(WINDOW *, int, attr_t, short, const void *);
int wclear(WINDOW *);
int wclrtobot(WINDOW *);
int wclrtoeol(WINDOW *);
void wcursyncup(WINDOW *);
int wcolor_set(WINDOW *, short, void *);
int wdelch(WINDOW *);
int wdeleteln(WINDOW *);
int wechochar(WINDOW *, const chtype);
int wecho_wchar(WINDOW *, const cchar_t *);
int werase(WINDOW *);
int wgetbkgrnd(WINDOW *, cchar_t *);
int wgetch(WINDOW *);
int wgetnstr(WINDOW *, const char *, int);
int wgetn_wstr(WINDOW *, wint_t *, int);
int wgetstr(WINDOW *, char *, int);
int wget_wstr(WINDOW *, wint_t *, int);
int whline(WINDOW *, chtype, int);
chtype winch(WINDOW *);
int winchnstr(WINDOW *, chtype *, int);
int winchstr(WINDOW *, chtype *);
int winnstr(WINDOW *, char *, int);
int winnwstr(WINDOW *, wchar_t *, int);
int winsch(WINDOW *, chtype);
int winsdelln(WINDOW *, int);
int insertln(WINDOW *);
int winsnstr(WINDOW *, const char *, int);
int wins_nwstr(WINDOW *, const wchar_t *, int);
int winsstr(WINDOW *, const char *);
int winstr(WINDOW *, char *);
int wins_wch(WINDOW *, const cchar_t *);
int wins_wstr(WINDOW *, const wchar_t *);
int win_wch(WINDOW *, cchar_t *);
int win_wchnstr(WINDOW *, cchar_t *, int);
int win_wchstr(WINDOW *, cchar_t *);
int winwstr(WINDOW *, wchar_t *);
int wmove(WINDOW *, int, int);
<curses.h>

int wnoutrefresh(WINDOW *);
int wprintw(WINDOW *, char *, ...);

int wredrawln(WINDOW *, int, int);

int wrefresh(WINDOW *);
int wscanw(WINDOW *, char *, ...);

int wscrl(WINDOW *, int);
int wsetscrreg(WINDOW *, int, int);
int wstandend(WINDOW *);
int wstandout(WINDOW *);

void wsyncup(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
Chapter 1, <stdio.h> (in the XSH specification), <term.h>, <termios.h> (in the XSH specification), <unctrl.h>, <wchar.h> (in the XSH specification).

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is completely rewritten to include new constants, data types and function prototypes.

Issue 4, Version 2
This entry is completely rewritten to correct the function prototypes.
NAME
term.h — terminal capabilities

SYNOPSIS
EC
#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 in the table in
Section 6.1.3 on page 241.

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);
int tputs(const char *, int, int (*)(int));

The <term.h> header defines the following data type through typedef:

bool As described in <curses.h>.

SEE ALSO
Chapter 6, printf(), putp(), tigetflag(), tgetent(), <curses.h>.

CHANGE HISTORY
First released in Issue 4.
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>.

CHANGE HISTORY

First released in Issue 4.
The requirements in this chapter are in effect only for implementations that claim Enhanced Curses compliance.

The terminfo 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 terminfo source files.

X/Open-compliant implementations must provide a facility that accepts source files in the format specified in this chapter as a means of entering information into the terminfo database. The facility for installing this information into the database is implementation-specific. A valid terminfo entry describing a given model of terminal can be added to terminfo on any X/Open-compliant implementation to permit use of the same terminal model.

Section 6.1 on page 238 describes the syntax of terminfo source files. A grammar and lexical conventions appear in Section 6.1.2 on page 239. A list of all terminal capabilities defined by X/Open appears in Section 6.1.3 on page 241. An example follows in Section 6.1.4 on page 251. Section A.1 on page 255 describes the specification of devices in general, such as video terminals. Section A.2 on page 268 describes the specification of printers.

The terminfo 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.
6.1 Source File Syntax

Source files can use the ISO 8859-1 codeset. The behaviour 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.

*terminfo* 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 *terminfo* source file must end in a comma. Every line in a *terminfo* source file except the header must be indented with one or more white spaces (either spaces or tabs).

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

```
alias_1 | alias_2 | ... | alias_n | longname,
<white space> am, lines #24,
<white space> home=\Eeh,
```

The first line, commonly referred to as the header line, must begin in column one and must contain at least two aliases separated by vertical bars. The last field in the header line must be the long name of the device and it may contain any string.

Alias names must be unique in the *terminfo* database and they must conform to file naming conventions established by implementation-specific *terminfo* compilation utilities. Implementations will recognise 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 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. Section A.4.1 on page 279 provides conventions for choosing alias names.

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>`.
6.1.1 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.

6.1.2 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_LINE4 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 ;

feature_lines : start_feature_line rest_of_feature_line |
               feature_lines start_feature_line rest_of_feature_line ;

start_feature_line : START_FEATURE_LINE_BOOLEAN5 |
                   START_FEATURE_LINE_NUMERIC6 |
                   START_FEATURE_LINE_STRING7 ;

rest_of_feature_line : features COMMA NEWLINE |
                      COMMA NEWLINE ;
```

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.
features : COMMA feature
  | features COMMA feature 
  ;

aliases : PIPE ALIAS
  | aliases PIPE ALIAS
  ;

feature : BOOLEAN
  | NUMERIC
  | STRING
  ;

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.
10. A feature line must not begin in column one.
11. Blank lines are ignored.

---

8. Graph characters are those characters for which `isgraph()` returns non-zero.
9. Print characters are those characters for which `isprint()` returns non-zero.
6.1.3 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() on page 194 (in the cases where the following table lists a Termcap code) and tigetflag() on page 196.

The table of capabilities has the following columns:

<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>cub1 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 colour</td>
</tr>
<tr>
<td>can_change</td>
<td>ccc</td>
<td>cc</td>
<td>Terminal can re-define existing colour</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 hpa/mhpa 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 cr turns off micro mode</td>
</tr>
<tr>
<td>dest_tabs_magic_smso</td>
<td>xt</td>
<td>xt</td>
<td>Destructive tabs, magic smso char (t1061)</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 &quot;status line&quot;</td>
</tr>
<tr>
<td>hue_lightness_saturation</td>
<td>his</td>
<td>h1</td>
<td>Terminal uses only HLS colour 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>
</tbody>
</table>
### Cap- Term- Description

<table>
<thead>
<tr>
<th>Variable</th>
<th>Cap-name</th>
<th>Term-cap</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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_rmocup</td>
<td>nrrmc</td>
<td>NR</td>
<td>smcup does not reverse rmcup</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>tilde_glitch</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>

### Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Cap-name</th>
<th>Term-cap</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>bufsz</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>
<tr>
<td>max_colors</td>
<td>colors</td>
<td>Co</td>
<td>Maximum number of colours on the screen</td>
</tr>
<tr>
<td>max_micro_address</td>
<td>maddr</td>
<td>Yd</td>
<td>Maximum value in micro_address</td>
</tr>
<tr>
<td>max_micro_jump</td>
<td>mjump</td>
<td>Ye</td>
<td>Maximum value in parm_.micro</td>
</tr>
<tr>
<td>max_pairs</td>
<td>pairs</td>
<td>pa</td>
<td>Maximum number of colour-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 colours</td>
</tr>
<tr>
<td>num_labels</td>
<td>nlab</td>
<td>N1</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>
</tbody>
</table>

X/Open CAE Specification 1996
<table>
<thead>
<tr>
<th>Variable</th>
<th>Cap-name</th>
<th>Term-cap</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>Cap-name</th>
<th>Term-cap</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>S8</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>bicr</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 ip 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>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>ci</td>
<td>Init sequence for multiple codesets</td>
</tr>
<tr>
<td>color_names</td>
<td>colornm</td>
<td>Yw</td>
<td>Give name for colour #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>Variable</td>
<td>Cap-name</td>
<td>Term-cap</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------</td>
<td>----------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cursor_mem_address</td>
<td>mrcup</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>cuf1</td>
<td>nd</td>
<td>Non-destructive space (cursor or carriage right)</td>
</tr>
<tr>
<td>cursor_to_ll</td>
<td>ll</td>
<td>ll</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>dl1</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>elhlm</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>enter_low_hl_mode</td>
<td>elohtm</td>
<td></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>ZK</td>
<td>Set near-letter quality print</td>
</tr>
<tr>
<td>enter_normal_quality</td>
<td>snrmq</td>
<td>ZL</td>
<td>Set normal quality print</td>
</tr>
<tr>
<td>enter_pc_charset_mode</td>
<td>smpch</td>
<td>S2</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></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>mk</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>
</tbody>
</table>

Source File Syntax
Terinfo Source Format (ENHANCED CURSES)

244 X/Open CAE Specification 1996
<table>
<thead>
<tr>
<th>Variable</th>
<th>Cap-name</th>
<th>Term-cap</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enter_subscript_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_hi_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_hi_mode</td>
<td>evhlm</td>
<td>SX</td>
<td>Turn on vertical highlight mode</td>
</tr>
<tr>
<td>enter_xon_mode</td>
<td>smxon</td>
<td>SX</td>
<td>Turn on xon/xoff handshake</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>ei</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>rmsec</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>rsubm</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 initialisation string</td>
</tr>
<tr>
<td>init_2string</td>
<td>is2</td>
<td>i2</td>
<td>Terminal or printer initialisation string</td>
</tr>
<tr>
<td>init_3string</td>
<td>is3</td>
<td>i3</td>
<td>Terminal or printer initialisation string</td>
</tr>
<tr>
<td>init_file</td>
<td>if</td>
<td>if</td>
<td>Name of initialisation file</td>
</tr>
<tr>
<td>init_prog</td>
<td>iprog</td>
<td>iP</td>
<td>Path name of program for initialisation</td>
</tr>
<tr>
<td>initialize_color</td>
<td>inic</td>
<td>IC</td>
<td>Set colour #1 to RGB #2, #3, #4</td>
</tr>
<tr>
<td>initialize_pair</td>
<td>initp</td>
<td>Ip</td>
<td>Set colour-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>al</td>
<td>Add new blank line</td>
</tr>
</tbody>
</table>
### Insert Padding

<table>
<thead>
<tr>
<th>Variable</th>
<th>Cap-name</th>
<th>Term-cap</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert_padding</td>
<td>ip</td>
<td>ip</td>
<td>Insert pad after character inserted</td>
</tr>
</tbody>
</table>

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()`).
<table>
<thead>
<tr>
<th>Variable</th>
<th>Cap- name</th>
<th>Term-cap</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key_mark</td>
<td>kmrk</td>
<td>%2</td>
<td>sent by mark key</td>
</tr>
<tr>
<td>key_message</td>
<td>kmsg</td>
<td>%3</td>
<td>sent by message key</td>
</tr>
<tr>
<td>key_mouse</td>
<td>kmous</td>
<td>Km</td>
<td>0631, Mouse event has occured</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>kN</td>
<td>sent by next-page key</td>
</tr>
<tr>
<td>key_open</td>
<td>kopn</td>
<td>%6</td>
<td>sent by open key</td>
</tr>
<tr>
<td>key_options</td>
<td>kopt</td>
<td>%7</td>
<td>sent by options key</td>
</tr>
<tr>
<td>key_ppage</td>
<td>kpp</td>
<td>KP</td>
<td>sent by previous-page key</td>
</tr>
<tr>
<td>key_previous</td>
<td>kprv</td>
<td>%8</td>
<td>sent by previous-object key</td>
</tr>
<tr>
<td>key_print</td>
<td>kprt</td>
<td>%9</td>
<td>sent by print or copy key</td>
</tr>
<tr>
<td>key_redo</td>
<td>krdo</td>
<td>%0</td>
<td>sent by redo key</td>
</tr>
<tr>
<td>key_reference</td>
<td>kref</td>
<td>&amp;1</td>
<td>sent by ref(ERENCE) key</td>
</tr>
<tr>
<td>key_refresh</td>
<td>krfr</td>
<td>%2</td>
<td>sent by refresh key</td>
</tr>
<tr>
<td>key_replace</td>
<td>krpl</td>
<td>%3</td>
<td>sent by replace key</td>
</tr>
<tr>
<td>key_restart</td>
<td>krst</td>
<td>%4</td>
<td>sent by restart key</td>
</tr>
<tr>
<td>key_resume</td>
<td>kres</td>
<td>%5</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>%6</td>
<td>sent by save key</td>
</tr>
<tr>
<td>key_sbeg</td>
<td>kBEG</td>
<td>%9</td>
<td>sent by shifted beginning key</td>
</tr>
<tr>
<td>key_scancel</td>
<td>kCAN</td>
<td>%0</td>
<td>sent by shifted cancel key</td>
</tr>
<tr>
<td>key_scommand</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>kFND</td>
<td>%0</td>
<td>sent by shifted find key</td>
</tr>
<tr>
<td>key_shelp</td>
<td>kHLP</td>
<td>%1</td>
<td>sent by shifted help key</td>
</tr>
<tr>
<td>key_shome</td>
<td>kHOM</td>
<td>%2</td>
<td>sent by shifted home key</td>
</tr>
<tr>
<td>key_sic</td>
<td>kIC</td>
<td>%3</td>
<td>sent by shifted input key</td>
</tr>
<tr>
<td>key_sleft</td>
<td>kLFT</td>
<td>%4</td>
<td>sent by shifted left-arrow key</td>
</tr>
<tr>
<td>key_smessage</td>
<td>kMSG</td>
<td>%a</td>
<td>sent by shifted message key</td>
</tr>
<tr>
<td>key_smove</td>
<td>kMOV</td>
<td>%b</td>
<td>sent by shifted move key</td>
</tr>
<tr>
<td>key_snxt</td>
<td>kNXT</td>
<td>%c</td>
<td>sent by shifted next key</td>
</tr>
<tr>
<td>key_soptions</td>
<td>kOPT</td>
<td>%d</td>
<td>sent by shifted options key</td>
</tr>
<tr>
<td>key_sprevious</td>
<td>kPRV</td>
<td>%e</td>
<td>sent by shifted prev key</td>
</tr>
<tr>
<td>key_sprint</td>
<td>kPRT</td>
<td>%f</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>%g</td>
<td>sent by shifted redo key</td>
</tr>
<tr>
<td>key_sreplace</td>
<td>kRPL</td>
<td>%h</td>
<td>sent by shifted replace key</td>
</tr>
<tr>
<td>key_sright</td>
<td>kRIT</td>
<td>%i</td>
<td>sent by shifted right-arrow key</td>
</tr>
<tr>
<td>Variable</td>
<td>Cap-name</td>
<td>Term-cap</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------</td>
<td>----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>key_resume</td>
<td>kRES</td>
<td>%j</td>
<td>sent by shifted resume key</td>
</tr>
<tr>
<td>key_save</td>
<td>kSAV</td>
<td>!1</td>
<td>sent by shifted save key</td>
</tr>
<tr>
<td>key_suspend</td>
<td>kSPD</td>
<td>!2</td>
<td>sent by shifted suspend key</td>
</tr>
<tr>
<td>key_tab</td>
<td>khts</td>
<td>kT</td>
<td>sent by set-tab key</td>
</tr>
<tr>
<td>key_undo</td>
<td>kUND</td>
<td>!3</td>
<td>sent by shifted undo key</td>
</tr>
<tr>
<td>key_suspend</td>
<td>kspd</td>
<td>&amp;7</td>
<td>sent by suspend key</td>
</tr>
<tr>
<td>key_undo</td>
<td>kund</td>
<td>&amp;8</td>
<td>sent by undo key</td>
</tr>
<tr>
<td>key_up</td>
<td>kc uu1</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>
<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>l00</td>
<td>10</td>
<td>Labels on function key f0 if not f0</td>
</tr>
<tr>
<td>lab_f1</td>
<td>l01</td>
<td>11</td>
<td>Labels on function key f1 if not f1</td>
</tr>
<tr>
<td>lab_f2</td>
<td>l02</td>
<td>12</td>
<td>Labels on function key f2 if not f2</td>
</tr>
<tr>
<td>lab_f3</td>
<td>l03</td>
<td>13</td>
<td>Labels on function key f3 if not f3</td>
</tr>
<tr>
<td>lab_f4</td>
<td>l04</td>
<td>14</td>
<td>Labels on function key f4 if not f4</td>
</tr>
<tr>
<td>lab_f5</td>
<td>l05</td>
<td>15</td>
<td>Labels on function key f5 if not f5</td>
</tr>
<tr>
<td>lab_f6</td>
<td>l06</td>
<td>16</td>
<td>Labels on function key f6 if not f6</td>
</tr>
<tr>
<td>lab_f7</td>
<td>l07</td>
<td>17</td>
<td>Labels on function key f7 if not f7</td>
</tr>
<tr>
<td>lab_f8</td>
<td>l08</td>
<td>18</td>
<td>Labels on function key f8 if not f8</td>
</tr>
<tr>
<td>lab_f9</td>
<td>l09</td>
<td>19</td>
<td>Labels on function key f9 if not f9</td>
</tr>
<tr>
<td>lab_f10</td>
<td>l010</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>LO</td>
<td>Turn on soft labels</td>
</tr>
<tr>
<td>meta_off</td>
<td>rmm</td>
<td>mo</td>
<td>Turn off &quot;meta mode&quot;</td>
</tr>
<tr>
<td>meta_on</td>
<td>smm</td>
<td>mm</td>
<td>Turn on &quot;meta mode&quot; (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>Z2</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>mc uu1</td>
<td>Zd</td>
<td>Like cursor_up for micro adjustment</td>
</tr>
<tr>
<td>mouse_info</td>
<td>m info</td>
<td>Mi</td>
<td>Mouse status information</td>
</tr>
<tr>
<td>newline</td>
<td>nel</td>
<td>nw</td>
<td>Newline (behaves like cr followed by lf)</td>
</tr>
<tr>
<td>order_of_pins</td>
<td>p order</td>
<td>Z e</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 colour(-pair)s to the original ones</td>
</tr>
<tr>
<td>orig_pair</td>
<td>op</td>
<td>op</td>
<td>Set default colour-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>d ch</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>Variable</td>
<td>Cap-name</td>
<td>Term-cap</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</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>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>Zt</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>p1</td>
<td>Prog funct key #1 to execute string #2</td>
</tr>
<tr>
<td>pkey_plab</td>
<td>pfxl</td>
<td>xl</td>
<td>Prog key #1 to xmit string #2 and show string #3</td>
</tr>
<tr>
<td>pkey_xmit</td>
<td>pf</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>mc4</td>
<td>pf</td>
<td>Turn off the printer</td>
</tr>
<tr>
<td>prtr_on</td>
<td>mc5</td>
<td>po</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>QD</td>
<td>Dial phone number #1, without progress detection</td>
</tr>
<tr>
<td>remove_clock</td>
<td>rmclik</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>RQ</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>scescal</td>
<td>S7</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>scs</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>sgr</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 colour to #1 using ANSI escape</td>
</tr>
<tr>
<td>set_a_foreground</td>
<td>setaf</td>
<td>AF</td>
<td>Set foreground colour to #1 using ANSI escape</td>
</tr>
<tr>
<td>set_a_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 colour 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>Variable</td>
<td>Cap-name</td>
<td>Term-cap</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</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 colour #1</td>
</tr>
<tr>
<td>set_color_pair</td>
<td>scp</td>
<td>sp</td>
<td>Set current colour pair to #1</td>
</tr>
<tr>
<td>set_foreground</td>
<td>setf</td>
<td>Sf</td>
<td>Set foreground colour 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>smgrl</td>
<td>ML</td>
<td>Sets both left and right margins</td>
</tr>
<tr>
<td>set_page_length</td>
<td>slines</td>
<td>YZ</td>
<td>Set page length to #1 lines</td>
</tr>
<tr>
<td>set_pglen_inch</td>
<td>slength</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_tab</td>
<td>hts</td>
<td>st</td>
<td>Set a tab in all rows, current column</td>
</tr>
<tr>
<td>set_tb_margin</td>
<td>smgtb</td>
<td>MT</td>
<td>Sets both top and bottom margins</td>
</tr>
<tr>
<td>set_top_margin</td>
<td>smgt</td>
<td>Zo</td>
<td>Set top margin at current line</td>
</tr>
<tr>
<td>set_top_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>subs</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>TO</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>
6.1.4 Sample Entry

The following entry describes the AT&T 610 terminal.

```
610 | 610bct | ATT610 | att610 | AT&T 610; 80 column; 98key keyboard,
am, esok, hs, mir, magr, xenl, xon,
cols#80, it#8, lh#2, lines#24, lw#8, nlab#8, ws1#80,
acsc="`aaffggjkkllmmnooppqqrrssttuuvvwwxxyyzz({|})~`",
bel=`G`, blink=`E[5m`, bold=`E[1m`, cbt=`E[Z`,
civis=`E]?2S`, clear=`E[H]E[J`, cnorm=`E[?25h`E[?12l,
cr=`\`, csr=`E[5p1d%p2%dr`, cub=`E[5p1d`, cubl=`b`,
cud=`E[5p1d`, cudl=`E[B`, cuuf=`E[5p1dC`, cuuf1=`E[C`,
cup=`E[5i%p1d%p2%dH`, cuu=`E[5p1%A`, cuul=`E[5p1%D`,
cvis=`E[12`25h`, dch=`E[5p1dF`, dchl=`E[P`, dim=`E[2m`,
dr=`E[5p1%M`, ed=`E[J`, el=`E[K`, el1=`E[1K`,
flash=`E[5h$<200>`E[5l`, fsl=`E[8`, home=`E[H]E[H]`, ht=`t`,
ich=`E[5p1%d`, il=`E[5p1%L`, il1=`E[5p1%L`,
inst=`E[8m`, is1=`E[0m`E(B`E)0`, is2=`E[0m`E(B`E)0`,
kbs=`H`, kcbt=`E[Z`, kclr=`E[2J`, kcu1=`E[D`, kcu1d=`E[B`,
kcf1=`E[C`, kcuul=`E[5A`, kP=`E0c`, kF0=`E0p`,
kP1=`E0g`, kP2=`E0r`, kP3=`E0s`, kF4=`E0t`,
kI=`E0d`, kB=`E0e`, kF4=`E0f`, kf=`E0g`,
kF6=`E0h`, kF7=`E0i`,
kF8=`E0j`, kF9=`E0k`, khome=`E[H]E[K]`,
kind=`E[S]`, kri=`E[T]`,
ll=`E[24h`, mc4=`E[4i`, mc5=`E[5i`, nel=`E[8E`,
pxfl=`E[5p1d%p2%dQ%?p1%(9)%<t`s`s`f1p`ld`s`s`s`
```

6.1.5 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, in situations described in Section A.1.16 on page 268.
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=$\text{EK}$<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-\text{x} (for any appropriate \text{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 (\text{^})</td>
</tr>
<tr>
<td>\</td>
<td>Backslash (\text{})</td>
</tr>
<tr>
<td>,</td>
<td>Comma (,)</td>
</tr>
<tr>
<td>:</td>
<td>Colon (:)</td>
</tr>
<tr>
<td>\0</td>
<td>Null</td>
</tr>
</tbody>
</table>
\nnnn     Any character, specified as three octal digits

(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 in the example in Section 6.1.4 on page 251. Note that capabilities are defined in a left-to-right order and, therefore, a prior definition will override a later definition.
A.1 Device Capabilities

A.1.1 Basic Capabilities

The number of columns on each line for the device is given by the `cols` numeric capability. If the device has a screen, then the number of lines on the screen is given by the `lines` capability. If the device wraps around to the beginning of the next line when it reaches the right margin, then it should have the `am` capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the `clear` string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over) then it should have the `os` capability. If the device is a printing terminal, with no soft copy unit, specify both `hc` and `os`. If there is a way to move the cursor to the left edge of the current row, specify this as `cr`. (Normally this will be carriage return, control-M.) If there is a way to produce an audible signal (such as a bell or a beep), specify it as `bel`. If, like most devices, the device uses the xon-xoff flow-control protocol, specify `xon`.

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

A very important point here is that the local cursor motions encoded in `terminfo` are undefined at the left and top edges of a screen terminal. Programs should never attempt to backspace around the left edge, unless `bw` is specified, and should never attempt to go up locally off the top. To scroll text up, a program goes to the bottom left corner of the screen and sends the `ind` (index) string. To scroll text down, a program goes to the top left corner of the screen and sends the `ri` (reverse index) string. The strings `ind` and `ri` are undefined when not on their respective corners of the screen.

Parameterised 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 and 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, initialisation strings should turn on this option, if possible. If the device has a command that moves to the first column of the next line, that command can be given as `nel` (newline). It does not matter if the command clears the remainder of the current line, so if the device has no `cr` and `lf` it may still be possible to craft a working `nel` out of one or both of them.

These capabilities suffice to describe hardcopy and screen terminals. Thus the AT&T 5320 hardcopy terminal is described as follows:
Device Capabilities

AT&T 5320 hardcopy terminal, cols#132, bel=^G, cr=^r, cubl=\b, cndl=\n, dchl=E[P, dll=E[M, ind=\n, ami, hc, os, am, bel=^G, cr=^M, cubl=^H, cudl=^J, ind=^J, lines#24,

while the Lear Siegler ADM-3 is described as

A.1.2 Parameterised Strings

Cursor addressing and other strings requiring arguments are described by a argumentised 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 ‘%’.
%[[I|flags][width][precision]][[doXs] 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 integer_2 op pop integer_1) where integer_1 represents the top of the stack.
%& %| %^ Bit operations: push(pop integer_2 op pop integer_1)
%>= %> %< Logical operations: push(pop integer_2 op pop integer_1)

256 X/Open CAE Specification 1996
Logical operations: and, or

Unary operations: push(op pop())

(For ANSI terminals) add 1 to the first argument (if one argument present), or first two arguments (if more than one argument present).

If-then-else, elseif part is optional; else-if's are possible ala Algol 68: if c then b else c else d

If the ‘‘-’’ flag is used with ‘‘%[doxs]’’, then a colon must be placed between the ‘‘%’’ and the ‘‘-’’ to differentiate the flag from the binary ‘‘-%’’ operator. For example: ‘‘%-16.16s’’.

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are zero-padded as two digits. Thus its \cup capability is:

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

The Micro-Term ACT-IV needs the current row and column sent preceded by a ‘‘T’, with the row and column simply encoded in binary:

\cup=\^T%p1%c%p2%c

Devices that use ‘‘%c’’ need to be able to backspace the cursor (\u1), and to move the cursor up one line on the screen (\u01). This is necessary because it is not always safe to transmit \n, ‘‘D, and ‘‘R, as the system may change or discard them. (The library functions dealing with terminfo set tty modes so that tabs are never expanded, so ‘‘T is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus:

\cup=\E=%p1%’s%’%c%p2%’s%’%c

After sending ‘‘\E=’’, this pushes the first argument, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values), and outputs that value as a character. Then the same is done for the second argument. More complex arithmetic is possible using the stack.

\section{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 \u1 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 argumentised local motions (such as \move n spaces to the right’’), these can be given as \u2, \u3, \u4, and \u5 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`.

### A.1.4 Area Clears

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

### A.1.5 Insert/Delete Line

If the terminal can open a new blank line before the line where the cursor is, this should be given as `il1`; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as `dl1`; this is done only from the first position on the line to be deleted. Versions of `il1` and `dl1` which take a single 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 argumentised 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.
A.1.6 Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character operations which can be described using terminfo. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin-Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type “abc def” using local cursor motions (not spaces) between the abc and the def. Then position the cursor before the abc and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the abc shifts over to the def which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability in, which stands for “insert null.” While these are two logically separate attributes (one line versus multiline insert mode, and special treatment of untyped spaces) we have seen no terminals whose insert mode cannot be described with the single attribute.

terminfo can describe both terminals that have an insert mode and terminals which send a simple sequence to open a blank position on the current line. Give as smir the sequence to get into insert mode. Give as rmir the sequence to leave insert mode. Now give as ich1 any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give ich1; terminals that send a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to ich1. Do not give both unless the terminal 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 mir to speed up inserting in this case. Omitting mir will affect only speed. Some terminals (notably Datamedia) must not have mir because of the way their insert mode works.

Finally, you can specify dch1 to delete a single character, dch with one argument, n, to delete n characters, and delete mode by giving smdc and rmdc to enter and exit delete mode (any mode the terminal needs to be placed in for dch1 to work).

A command to erase n characters (equivalent to outputting n blanks without moving the cursor) can be given as ech with one argument.
A.1.7 Highlighting, Underlining and Visible Bells

Your device may have one or more kinds of display attributes that allow you to highlight selected characters when they appear on the screen. The following display modes (shown with the names by which they are set) may be available:

- A blinking screen (blink)
- Bold or extra-bright characters (bold)
- Dim or half-bright characters (dim)
- Blanking or invisible text (invis)
- Protected text (prot)
- A reverse-video screen (rev)
- An alternate character set (smacs to enter this mode and rmacs to exit it) (If a command is necessary before you can enter alternate character set mode, give the sequence in enacs or "enable alternate-character-set” mode.) Turning on any of these modes singly may turn off other modes.

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

Choose one display method as 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 smso and rmso, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then xmc should be given to tell how many spaces are left.

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

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

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), then this can be given as flash; it must not move the cursor. A good flash can be done by changing the screen into reverse video, pad for 200 ms, then return the screen to normal video.

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

If your terminal generates underlined characters by using the underline character (with no special sequences needed) even though it does not otherwise overstrike characters, then 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>tparm</th>
<th>Argument</th>
<th>Attribute</th>
<th>Escape Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td></td>
<td></td>
<td>\E[0m</td>
</tr>
<tr>
<td>p1</td>
<td>standout</td>
<td></td>
<td>\E[0;4;7m</td>
</tr>
<tr>
<td>p2</td>
<td>underline</td>
<td></td>
<td>\E[0;3m</td>
</tr>
<tr>
<td>p3</td>
<td>reverse</td>
<td></td>
<td>\E[0;4m</td>
</tr>
<tr>
<td>p4</td>
<td>blink</td>
<td></td>
<td>\E[0;5m</td>
</tr>
<tr>
<td>p5</td>
<td>dim</td>
<td></td>
<td>\E[0;7m</td>
</tr>
<tr>
<td>p6</td>
<td>bold</td>
<td></td>
<td>\E[0;3;4m</td>
</tr>
<tr>
<td>p7</td>
<td>invis</td>
<td></td>
<td>\E[0;8m</td>
</tr>
<tr>
<td>p8</td>
<td>protect</td>
<td></td>
<td>not available</td>
</tr>
<tr>
<td>p9</td>
<td>altcharset</td>
<td></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[03;5m`. The terminal doesn’t have `protect` mode, either, but that cannot be simulated in any way, so `p8` is ignored. The `altcharset` mode is different in that it is either `^O` or `^N`, depending on whether it is off or on. If all modes were to be turned on, the sequence would be:

`\E[0;3;4;5;7;8m^N`

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

<table>
<thead>
<tr>
<th>Sequence</th>
<th>When to Output</th>
<th>tparm Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\E[0</td>
<td>always</td>
<td>\E[0</td>
</tr>
<tr>
<td>;3</td>
<td>if p2 or p6</td>
<td>%?%p2%p6%</td>
</tr>
<tr>
<td>;4</td>
<td>if p1 or p3 or p6</td>
<td>%?%p1%p3%</td>
</tr>
<tr>
<td>;5</td>
<td>if p4</td>
<td>%?%p4%t;5%;</td>
</tr>
<tr>
<td>;7</td>
<td>if p1 or p5</td>
<td>%?%p1%p5%</td>
</tr>
<tr>
<td>;8</td>
<td>if p7</td>
<td>%?%p7%t;8%;</td>
</tr>
<tr>
<td>m</td>
<td>always</td>
<td>m</td>
</tr>
<tr>
<td>^N or ^O</td>
<td>if p9 ^N, else ^O</td>
<td>%?%p9%t ^N%e ^O%</td>
</tr>
</tbody>
</table>

Putting this all together into the `sgr` sequence gives:

`sgr=\E[0?%p2%p6% | t;3%; %?%p1%p3% | %p6% | t;4%; %?%p5%t;5%; %?%p1%p5% | t;7%; %?%p7%t;8%; m?%p9%t ^N%e ^O%; ,`
A.1.8 Keypad

If the device has a keypad that transmits sequences when the keys are pressed, this information can also be specified. Note that it is not possible to handle devices where the keypad only works in local (this applies, for example, to the unshifted Hewlett-Packard 2621 keys). If the keypad can be set to transmit or not transmit, specify these sequences as smlx and rmkx. Otherwise the keypad is assumed to always transmit.

The sequences sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as kcu1, kcuf1, kceu1, kcuu1 and khome, respectively. If there are function keys such as f0, f1, ..., f63, the sequences they send can be specified as kff0, kff1, ..., kff63. If the first 11 keys have labels other than the default f0 through f10, the labels can be given as kf0, kf1, ..., kf63. These keys are useful when the effects of a 3 by 3 directional pad are needed. Further keys are defined above in the capabilities list.

Strings to program function keys can be specified as pfkey, pfloc, and pf. A string to program screen labels should be specified as pln. Each of these strings takes two arguments: a function key identifier and a string to program it with. pfkey causes pressing the given key to be the same as the user typing the given string; pfloc causes the string to be executed by the terminal in local mode; and pf causes the string to be transmitted to the computer. The capabilities nlab, lw and lh define the number of programmable screen labels and their width and height. If there are commands to turn the labels on and off, give them in smln and rmln. Smln is normally output after one or more pln sequences to make sure that the change becomes visible.

A.1.9 Tabs and Initialisation

If the device has hardware tabs, the command to advance to the next tab stop can be given as ht (usually control-I). A ‘backtab’ command that moves leftward to the next tab stop can be given as cbt. By convention, if tty modes show that tabs are being expanded by the computer rather than being sent to the device, programs should not use ht or cbt (even if they are present) because the user 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 n is given, showing the number of spaces the tabs are set to. This is normally used by 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 terminfo description can assume that they are properly set. If there are commands to set and clear tab stops, they can be given as tbc (clear all tab stops) and hts (set a tab stop in the current column of every row).

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

This is usually done using the init option of tput.
Most initialisation 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 initialisation 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, see \texttt{Margins} on page 273.)

### A.1.10 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}.

### A.1.11 Status Lines

If the terminal has an extra “status line” that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, into which one can cursor address normally (such as the Heathkit H19’s 25th line, or the 24th line of a VT100 which is set to a 23-line scrolling region), the capability \texttt{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 \texttt{tsl} and \texttt{fsl}. (\texttt{fsl} must leave the cursor position in the same place it was before \texttt{tsl}. If necessary, the \texttt{sc} and \texttt{rc} strings can be included in \texttt{tsl} and \texttt{fsl} to get this effect.) The capability \texttt{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 \texttt{tab}, work while in the status line, the flag \texttt{eslok} can be given. A string which turns off the status line (or otherwise erases its contents) should be given as \texttt{dsl}. If the terminal has commands to save and restore the position of the cursor, give them as \texttt{sc} and \texttt{rc}. The status line is normally assumed to be the same width as the rest of the screen (that is, \texttt{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 \texttt{wsl}.
A.1.12  Line Graphics

If the device has a line drawing alternate character set, the mapping of glyph to character would be given in \texttt{asc}. 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 (\textbackslash{ })</td>
<td>t</td>
</tr>
<tr>
<td>right tee (\textbackslash{-})</td>
<td>u</td>
</tr>
<tr>
<td>bottom tee (\textbackslash{\textbackslash{ ) } }</td>
<td>v</td>
</tr>
<tr>
<td>top tee (\textbackslash{\textbackslash{T} } )</td>
<td>w</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:

<table>
<thead>
<tr>
<th>Glyph Name</th>
<th>VT100+ Character</th>
<th>Character Used on New Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper left corner</td>
<td>l</td>
<td>R</td>
</tr>
<tr>
<td>lower left corner</td>
<td>m</td>
<td>F</td>
</tr>
<tr>
<td>upper right corner</td>
<td>k</td>
<td>T</td>
</tr>
<tr>
<td>lower right corner</td>
<td>j</td>
<td>G</td>
</tr>
<tr>
<td>horizontal line</td>
<td>q</td>
<td>,</td>
</tr>
<tr>
<td>vertical line</td>
<td>x</td>
<td>.</td>
</tr>
</tbody>
</table>

Now write down the characters left to right; for example:
\texttt{acsc=1RmFkTjGq\,x.}

In addition, \texttt{terminfo} lets you define multiple character sets (see Section A.2.5 on page 275).
A.1.13 Colour Manipulation

Most colour terminals belong to one of two classes of terminal:

- Tektronix-style

  The Tektronix method uses a set of \( N \) predefined colours (usually 8) from which an application can select "current" foreground and background colours. Thus a terminal can support up to \( N \) colours mixed into \( N \times N \) colour-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 colour-pair at once. Up to \( M \) colour-pairs, made from \( 2^M \) different colours, can be defined this way.

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

Tektronix 4100 series terminals only use HLS colour 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 colours, but uses a colour 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 colour. By convention, the first eight colours (0–7) map to, in order: black, red, green, yellow, blue, magenta, cyan, white. However, colour re-mapping may occur or the underlying hardware may not support these colours. Mappings for any additional colours supported by the device (that is, to numbers greater than 7) are at the discretion of the `terminfo` entry writer.

To initialise a colour-pair (HP method), use `initp` (initialise pair). It requires seven arguments: the number of a colour-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 colour-pair current, use `scp` (set colour-pair). It takes one argument, the number of a colour-pair.

Some terminals (for example, most colour terminal emulators for PCs) erase areas of the screen with current background colour. In such cases, `bce` (background colour erase) should be defined. The variable `op` (original pair) contains a sequence for setting the foreground and the background colours to what they were at the terminal start-up time. Similarly, `oc` (original colours) contains a control sequence for setting all colours (for the Tektronix method) or colour-pairs (for the HP method) to the values they had at the terminal start-up time.
Some colour terminals substitute colour for video attributes. Such video attributes should not be combined with colours. Information about these video attributes should be packed into the \texttt{ncv} (no colour 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_STANDARD</td>
<td>0</td>
<td>1</td>
<td>\texttt{sgr}, parameter \texttt{1}</td>
</tr>
<tr>
<td>WA_UNDERLINE</td>
<td>1</td>
<td>2</td>
<td>\texttt{sgr}, parameter \texttt{2}</td>
</tr>
<tr>
<td>WA_REVERSE</td>
<td>2</td>
<td>4</td>
<td>\texttt{sgr}, parameter \texttt{3}</td>
</tr>
<tr>
<td>WA_BLINK</td>
<td>3</td>
<td>8</td>
<td>\texttt{sgr}, parameter \texttt{4}</td>
</tr>
<tr>
<td>WA_DIM</td>
<td>4</td>
<td>16</td>
<td>\texttt{sgr}, parameter \texttt{5}</td>
</tr>
<tr>
<td>WA_BOLD</td>
<td>5</td>
<td>32</td>
<td>\texttt{sgr}, parameter \texttt{6}</td>
</tr>
<tr>
<td>WA_INVIS</td>
<td>6</td>
<td>64</td>
<td>\texttt{sgr}, parameter \texttt{7}</td>
</tr>
<tr>
<td>WA_PROTECT</td>
<td>7</td>
<td>128</td>
<td>\texttt{sgr}, parameter \texttt{8}</td>
</tr>
<tr>
<td>WA_ALTCARSET</td>
<td>8</td>
<td>256</td>
<td>\texttt{sgr}, parameter \texttt{9}</td>
</tr>
<tr>
<td>WA_HORIZONTAL</td>
<td>9</td>
<td>512</td>
<td>\texttt{sgr1}, parameter \texttt{1}</td>
</tr>
<tr>
<td>WA_LEFT</td>
<td>10</td>
<td>1024</td>
<td>\texttt{sgr1}, parameter \texttt{2}</td>
</tr>
<tr>
<td>WA_LOW</td>
<td>11</td>
<td>2048</td>
<td>\texttt{sgr1}, parameter \texttt{3}</td>
</tr>
<tr>
<td>WA_RIGHT</td>
<td>12</td>
<td>4096</td>
<td>\texttt{sgr1}, parameter \texttt{4}</td>
</tr>
<tr>
<td>WA_TOP</td>
<td>13</td>
<td>8192</td>
<td>\texttt{sgr1}, parameter \texttt{5}</td>
</tr>
<tr>
<td>WA_VERTICAL</td>
<td>14</td>
<td>16384</td>
<td>\texttt{sgr1}, parameter \texttt{6}</td>
</tr>
</tbody>
</table>

When a particular video attribute should not be used with colours, set the corresponding \texttt{ncv} bit to 1; otherwise set it to 0. To determine the information to pack into the \texttt{ncv} variable, add the decimal values corresponding to those attributes that cannot coexist with colours. For example, if the terminal uses colours to simulate reverse video (bit number 2 and decimal value 4) and bold (bit number 5 and decimal value 32), the resulting value for \texttt{ncv} will be 36 (4 + 32).

\section*{A.1.14 Miscellaneous}

If the terminal requires other than a null (zero) character as a pad, then this can be given as \texttt{pad}. Only the first character of the \texttt{pad} string is used. If the terminal does not have a pad character, specify \texttt{npc}.

If the terminal can move up or down half a line, this can be indicated with \texttt{hu} (half-line up) and \texttt{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 \texttt{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 argumentised string \texttt{rep}. The first argument is the character to be repeated and the second is the number of times to repeat it. Thus, \texttt{tparm(\texttt{repeat_char}, 'x', 10)} is the same as \texttt{xxxxxxx}.

If the terminal has a settable command character, such as the Tektronix 4025, this can be indicated with \texttt{cmdch}. A prototype command character is chosen which is used in all capabilities. This character is given in the \texttt{cmdch} capability to identify it. The following convention is supported on some systems: If the environment variable \texttt{CC} exists, all occurrences of the prototype character are replaced with the character in \texttt{CC}.

Terminal descriptions that do not represent a specific kind of known terminal, such as \texttt{switch}, \texttt{dialup}, \texttt{patch}, and \texttt{network}, should include the \texttt{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 sxmxon 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 mc5i (silent printer). All text, including mc4, is transparently passed to the printer while an mc5p is in effect.

A.1.15 Special Cases

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

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

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

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

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

For Beehive Superbee terminals that do not transmit the escape or control-C characters, specify xsb, indicating that the f1 key is to be used for escape and the f2 key for control-C.
A.1.16 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.

A.2 Printer Capabilities

The terminfo database lets you define capabilities of printers as well as terminals. Capabilities available for printers are included in the lists in Section 6.1.3 on page 241.

A.2.1 Rounding Values

Because argumentised 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 argumentised string capability.

A.2.2 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.

A.2.3 Specifying Printer Resolution

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

<table>
<thead>
<tr>
<th>Characteristic Number of Smallest Steps</th>
<th>Description</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.

<table>
<thead>
<tr>
<th>Automatic Motion after Printing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Normal Mode:</strong></td>
</tr>
<tr>
<td>orc</td>
</tr>
<tr>
<td>orl</td>
</tr>
<tr>
<td><strong>Micro Mode:</strong></td>
</tr>
<tr>
<td>mcs</td>
</tr>
<tr>
<td>mls</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.

<table>
<thead>
<tr>
<th>Automatic Motion after Printing Wide Character</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Normal Mode or Micro Mode (mcs = orc):</strong></td>
</tr>
<tr>
<td>widcs</td>
</tr>
<tr>
<td><strong>Micro Mode (mcs &lt; orc):</strong></td>
</tr>
<tr>
<td>mcs</td>
</tr>
</tbody>
</table>
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>Control Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cpi</code></td>
<td>Change character pitch</td>
</tr>
<tr>
<td><code>cpix</code></td>
<td>If set, <code>cpi</code> changes <code>orhi</code>, otherwise changes <code>orc</code></td>
</tr>
<tr>
<td><code>lpi</code></td>
<td>Change line pitch</td>
</tr>
<tr>
<td><code>lpix</code></td>
<td>If set, <code>lpi</code> changes <code>orvi</code>, otherwise changes <code>orl</code></td>
</tr>
<tr>
<td><code>chr</code></td>
<td>Change steps per column</td>
</tr>
<tr>
<td><code>cvr</code></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; see Section A.2.7 on page 277).

### Effects of Changing the Character/Line Pitches

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using <code>cpi</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>with <code>cpix</code> clear:</td>
<td><code>orhi</code></td>
<td><code>orhi</code></td>
</tr>
<tr>
<td><code>orci</code></td>
<td></td>
<td><code>orc</code></td>
</tr>
<tr>
<td></td>
<td><code>V_{cpi}</code></td>
<td></td>
</tr>
<tr>
<td>Using <code>cpi</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>with <code>cpix</code> set:</td>
<td><code>orhi</code> = <code>orc</code> $V_{cpi}$</td>
<td></td>
</tr>
<tr>
<td><code>orci</code></td>
<td></td>
<td><code>orc</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Using <code>lpi</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>with <code>lpix</code> clear:</td>
<td><code>orvi</code></td>
<td><code>orvi</code></td>
</tr>
<tr>
<td><code>orvi</code></td>
<td></td>
<td><code>orvi</code></td>
</tr>
<tr>
<td></td>
<td><code>V_{lpix}</code></td>
<td></td>
</tr>
<tr>
<td>Using <code>lpi</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>with <code>lpix</code> set:</td>
<td><code>orvi</code> = <code>orvi</code> $V_{lpix}$</td>
<td></td>
</tr>
<tr>
<td><code>orvi</code></td>
<td></td>
<td><code>orvi</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Using <code>chr</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>orhi</code></td>
<td></td>
<td><code>orhi</code></td>
</tr>
<tr>
<td><code>orc</code></td>
<td></td>
<td>$V_{chr}$</td>
</tr>
<tr>
<td>Using <code>cvr</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>orvi</code></td>
<td></td>
<td><code>orvi</code></td>
</tr>
<tr>
<td><code>orvi</code></td>
<td></td>
<td>$V_{cvr}$</td>
</tr>
</tbody>
</table>
A.2.4 Capabilities that Cause Movement

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

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

### String Capabilities for Motion

<table>
<thead>
<tr>
<th>String</th>
<th>Description</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.

### Limits to Motion

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mjump</td>
<td>Limit on use of mcub1, mcuf1, mcuu1, mcud1</td>
</tr>
<tr>
<td>maddr</td>
<td>Limit on use of mhpa, mvpa</td>
</tr>
<tr>
<td>xhpa</td>
<td>If set, hpa and mhpa can’t move left</td>
</tr>
<tr>
<td>xvpa</td>
<td>If set, vpa and mvpa 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.
**Entering/Exiting Micro Mode**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</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.

**What Happens After Character Printed in Rightmost Position**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sam</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.

**Entering/Exiting Reverse Modes**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slm</td>
<td>Reverse sense of horizontal motions</td>
</tr>
<tr>
<td>rlm</td>
<td>Restore sense of horizontal motions</td>
</tr>
<tr>
<td>sum</td>
<td>Reverse sense of vertical motions</td>
</tr>
<tr>
<td>rum</td>
<td>Restore sense of vertical motions</td>
</tr>
</tbody>
</table>

*While sense of horizontal motions reversed:*

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcub1</td>
<td>Move 1 step right</td>
</tr>
<tr>
<td>mcuf1</td>
<td>Move 1 step left</td>
</tr>
<tr>
<td>mcub</td>
<td>Move N steps right</td>
</tr>
<tr>
<td>mcuf</td>
<td>Move N steps left</td>
</tr>
<tr>
<td>cub1</td>
<td>Move 1 column right</td>
</tr>
<tr>
<td>cuf1</td>
<td>Move 1 column left</td>
</tr>
<tr>
<td>cub</td>
<td>Move N columns right</td>
</tr>
<tr>
<td>cuf</td>
<td>Move N columns left</td>
</tr>
</tbody>
</table>

*While sense of vertical motions reversed:*

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcuu1</td>
<td>Move 1 step down</td>
</tr>
<tr>
<td>mcud1</td>
<td>Move 1 step up</td>
</tr>
<tr>
<td>mcuu</td>
<td>Move N steps down</td>
</tr>
<tr>
<td>mcud</td>
<td>Move N steps up</td>
</tr>
<tr>
<td>cuu1</td>
<td>Move 1 line down</td>
</tr>
<tr>
<td>cud1</td>
<td>Move 1 line up</td>
</tr>
<tr>
<td>cuu</td>
<td>Move N lines down</td>
</tr>
<tr>
<td>cud</td>
<td>Move N lines up</td>
</tr>
</tbody>
</table>

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.

### Miscellaneous Motion Strings

<table>
<thead>
<tr>
<th>docr</th>
<th>List of control characters causing cr</th>
</tr>
</thead>
<tbody>
<tr>
<td>zerom</td>
<td>Prevent auto motion after printing next single character</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.

### Setting Margins

<table>
<thead>
<tr>
<th>smgl</th>
<th>Set left margin at current column</th>
</tr>
</thead>
<tbody>
<tr>
<td>smgr</td>
<td>Set right margin at current column</td>
</tr>
<tr>
<td>smgb</td>
<td>Set bottom margin at current line</td>
</tr>
<tr>
<td>smgt</td>
<td>Set top margin at current line</td>
</tr>
<tr>
<td>smgbp</td>
<td>Set bottom margin at line N</td>
</tr>
<tr>
<td>smglp</td>
<td>Set left margin at column N</td>
</tr>
<tr>
<td>smgrp</td>
<td>Set right margin at column N</td>
</tr>
<tr>
<td>smgtp</td>
<td>Set top margin at line N</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 *smglp* and *smgrp* or *smgtp* and *smgbp* should be defined; the other should be left blank. When writing an application that uses these string capabilities, the pairs should be first checked to see if each in the pair is set or only one is set, and should then be used accordingly.
In counting lines or columns, line zero is the top line and column zero is the left-most column. A zero value for the second argument with `smbp` means the bottom line of the page.

All margins can be cleared with `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>
</tr>
</thead>
<tbody>
<tr>
<td>sshm</td>
</tr>
<tr>
<td>rshm</td>
</tr>
<tr>
<td>sitm</td>
</tr>
<tr>
<td>ritm</td>
</tr>
<tr>
<td>swidm</td>
</tr>
<tr>
<td>rwidm</td>
</tr>
<tr>
<td>ssupm</td>
</tr>
<tr>
<td>rsupm</td>
</tr>
<tr>
<td>supcs</td>
</tr>
<tr>
<td>ssubm</td>
</tr>
<tr>
<td>rsubm</td>
</tr>
<tr>
<td>subcs</td>
</tr>
</tbody>
</table>

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

*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 italicised subscripts.

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

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

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

```
Bi  B_i  B^i
```

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

### A.2.5 Alternate Character Sets

In addition to allowing you to define line graphics (described in Section A.1.12 on page 264), `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>Control Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>scs</code> N</td>
<td>Select character set <code>N</code></td>
</tr>
<tr>
<td><code>scsd</code> N, M</td>
<td>Start definition of character set <code>N</code>, <code>M</code> characters</td>
</tr>
<tr>
<td><code>defc</code> A, B, D</td>
<td>Define character <code>A</code>, <code>B</code> dots wide, descender <code>D</code></td>
</tr>
<tr>
<td><code>rcsd</code> N</td>
<td>End definition of character set <code>N</code></td>
</tr>
<tr>
<td><code>csmn</code></td>
<td>List of character set names</td>
</tr>
<tr>
<td><code>daisy</code></td>
<td>Printer has manually changed print-wheels</td>
</tr>
</tbody>
</table>

The `scs`, `rcsd`, and `csmn` 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 initialised. 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; the number of these bytes and their form are defined in Section A.2.6 on page 276.

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 `csmn` string alleviates this problem by providing names for each number.

When used with a character set number in a call to `tparm`, the `csmn` 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
\texttt{csnm} string to determine the correct number), or by name, where the application examines the
\texttt{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 \texttt{daisy} is set.

A.2.6 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.

\begin{center}
\begin{tabular}{ll}
\textbf{Dot-Matrix Graphics} & \\
\texttt{npins} & Number of pins, \(N\), in print-head \\
\texttt{spinv} & Spacing of pins vertically in pins per inch \\
\texttt{spinh} & Spacing of dots horizontally in dots per inch \\
\texttt{porder} & Matches software bits to print-head pins \\
\texttt{sbim} & Start printing bit image graphics, \(B\) bits wide \\
\texttt{rbim} & End printing bit image graphics
\end{tabular}
\end{center}

The \texttt{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 \texttt{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
\texttt{npins} numeric capability. The size of the rectangle in fractions of an inch will also vary; it can be
deduced from the \texttt{spinv} and \texttt{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 \texttt{sbim} and \texttt{rbim} strings start and end a dot-matrix image, respectively. The \texttt{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 \texttt{sbim} string and before the \texttt{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 \texttt{porder} string as described below.

The \texttt{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 \texttt{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 \texttt{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 \texttt{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 \texttt{porder}, a 0 bit is used. If a position has a lower
case \texttt{'x'} instead of a pin number, a 1 bit is used in the skipped position. For consistency, a lower
case \texttt{'o'} can be used to represent a 0 filled, skipped bit. There must be a multiple of 8 bit
positions used or skipped in \texttt{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 \texttt{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 \texttt{porder} strings for these printers would be \texttt{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 \texttt{porder} strings for these printers would be \texttt{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 \texttt{porder} string for these printers would be \texttt{,6,5,4,3,2,1;63}, or alternately \texttt{o,o,6,5,4,3,2,1;63}.

\subsection*{A.2.7 Effect of Changing Printing Resolution}

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

\begin{center}
\begin{tabular}{ll}
\hline
\textbf{Changing the Character/Line Pitches} \\
\texttt{cpi} & Change character pitch \\
\texttt{cpix} & If set, \texttt{cpi} changes \texttt{spinh} \\
\texttt{lpi} & Change line pitch \\
\texttt{lpix} & If set, \texttt{lpi} changes \texttt{spinv} \\
\hline
\end{tabular}
\end{center}

Programs that use \texttt{cpi} or \texttt{lpi} should recalculate the dot spacing:

\begin{center}
\begin{tabular}{ll}
\hline
\textbf{Effects of Changing the Character/Line Pitches} \\
Before & After \\
\hline
Using \texttt{cpi} with \texttt{cpix} clear: & \\
\texttt{spinh}' & \texttt{spinh} \\
Using \texttt{cpi} with \texttt{cpix} set: & \\
\texttt{spinh}' & \texttt{spinh}=$\texttt{spinh}' \cdot \texttt{orhi}'$ \\
Using \texttt{lpi} with \texttt{lpix} clear: & \\
\texttt{spinv}' & \texttt{spinv} \\
Using \texttt{lpi} with \texttt{lpix} set: & \\
\texttt{spinv}' & \texttt{spinv}=$\texttt{spinv}' \cdot \texttt{orhi}'$ \\
Using \texttt{chr}: & \\
\texttt{spinh}' & \texttt{spinh} \\
Using \texttt{cvr}: & \\
\texttt{spinv}' & \texttt{spinv} \\
\hline
\end{tabular}
\end{center}

\texttt{orhi}' and \texttt{orhi} are the values of the horizontal resolution in steps per inch, before using \texttt{cpi} and after using \texttt{cpi}, respectively. Likewise, \texttt{orvi}' and \texttt{orvi} are the values of the vertical resolution in steps per inch, before using \texttt{lpi} and after using \texttt{lpi}, respectively. Thus, the changes in the dots per inch for dot-matrix graphics follow the changes in steps per inch for printer resolution.
A.2.8 Print Quality

Many dot-matrix printers can alter the dot spacing of printed text to produce near-letter-quality printing or draft-quality printing. 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>
</tr>
</thead>
<tbody>
<tr>
<td>snlq</td>
</tr>
<tr>
<td>snrmq</td>
</tr>
<tr>
<td>sdrfq</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.

A.2.9 Printing Rate and Buffer Size

Because there is no standard protocol that can be used to keep a program synchronised 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>
</tr>
</thead>
<tbody>
<tr>
<td>cps</td>
</tr>
<tr>
<td>bufsz</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 synchronise 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 recognise 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 favour of the user, who wants, above all, to see all the output in its correct place.
A.3 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.

A.4 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.

A.4.1 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:
### A.4.2 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 optimisations within Curses and thus operate more efficiently. For most terminals, an optimal `terminfo` entry has already been written.

<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>
<tr>
<td>-nam</td>
<td>Without automatic margins</td>
<td>vt100-nam</td>
</tr>
<tr>
<td>-n</td>
<td>Number of lines on the screen</td>
<td>2300-40</td>
</tr>
<tr>
<td>-na</td>
<td>No arrow keys (leave them in local)</td>
<td>c100-na</td>
</tr>
<tr>
<td>-np</td>
<td>Number of pages of memory</td>
<td>c100-4p</td>
</tr>
<tr>
<td>-rv</td>
<td>Reverse video</td>
<td>4415-rv</td>
</tr>
</tbody>
</table>
background
A property of a window that specifies a character (the background character) and a rendition to
be used in a variety of situations. See Section 3.3.6 on page 18.

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 colour 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

-w suffix ................................................................. 279
<curses.h> .............................................................. 220
<term.h> ................................................................. 235
<stdio.h> ............................................................... 236
@ ........................................................................... 268
XBD specification
  relationship to .................................................. 13
_w infix ................................................................... 26
_XOPEN_SOURCE .................................................. 9
asci ......................................................................... 264
add
  effect on straddling character ....................... 21
  resulting rendition ........................................... 22
add function ....................................................... 19
addch() ................................................................. 32
addchnstr() ......................................................... 33
addcstr() ............................................................. 33
addnstr() .............................................................. 34
addnstr() ............................................................. 35
addstr() ............................................................... 34
addwstr() ............................................................. 35
add_wch() ............................................................. 36
add_wchnstr() ..................................................... 37
add_wchstr() ....................................................... 37
adjustment of cursor position ......................... 20
advertised print rate ......................................... 278
advisory delay ...................................................... 252
alias
  in terminfo ....................................................... 238
alternate character set .................................... 260, 275
  line drawing ...................................................... 264
alternate keypad ............................................... 262
am ................................................................. 255
  ignoring linefeed after .................................... 267
ancestor ............................................................. 14
Ann Arbor 4080 (example) ................................ 257
ANSI foreground/background ............................ 265
ANSI X3.64-1979 ................................................... 238
application consideration .................................. 30
area clear ............................................................ 258
arrow keys .......................................................... 262
asterisk
  in terminfo ....................................................... 252
AT&T 4410v1
  line drawing ...................................................... 264
AT&T 470/475 ........................................................ 277
AT&T 5320 (example) ......................................... 255
AT&T 610 (example) ......................................... 251
attribute ............................................................. 16
attron() ............................................................... 38
attroff() .............................................................. 38
attrset() ............................................................. 38
attr_get() ........................................................... 39
attr_off() ........................................................... 39
attr_on() ............................................................ 39
attroff() ............................................................. 39
audible signal ..................................................... 255
automatic margin ............................................. 255
automatic motion .............................................. 274
auxiliary printer control .................................... 267
background ....................................................... 18, 281
background character ....................................... 18
  implicit use ....................................................... 22
background colour ............................................ 265
backslash
  use in terminfo ............................................... 238
backslash in terminfo ....................................... 252
backspace
  special processing .......................................... 21
  basic capability .............................................. 255
baud rate, versus printer throughput ............... 278
baudrate() ........................................................ 41
bce ................................................................. 265
Beehive Superbee ............................................. 267
beep() ................................................................. 42
bel ................................................................. 255
  delays ............................................................. 252
bell ................................................................. 255
  visible ........................................................... 260
bidirectional writing ........................................ 2
bkgd() ................................................................. 43
bkgrdset() ........................................................ 43
bkgrnd() ........................................................... 44
bkgrndset() ........................................................ 44
blinking text ..................................................... 260
blinking screen ................................................ 260
block cursor ...................................................... 260
block mode ....................................................... 30
bold ............................................................... 260
  printing ......................................................... 274
 boolean capability .......................................... 238
Index

cud ................................................................. 257
cuf ................................................................. 257
cuf1 ............................................................... 255
cup ................................................................. 256
current or specified position ....................... 26
current or specified window ....................... 26
current position ........................................... 271
curscr ........................................................... 63
Curses ......................................................... 1
Curses window ............................................. 281
cursor
actual position .............................................. 19
analogue in printing terminal .................... 271
appearance of .............................................. 260
cursor addressing ........................................ 256
cursor movement ......................................... 255
relocation .................................................... 20
within row or column .................................. 257
cursor position ............................................ 19, 281
at insert/delete ......................................... 20
curs_set() .................................................... 64
cur_term ...................................................... 65
cuu ............................................................... 257
cuu1 ............................................................. 257
cu[b/d/f/u][1] ................................................ 272
cvr ............................................................... 270
recalculate resolution after ....................... 270
cvvis ........................................................... 260
da ............................................................... 258
daisy ........................................................... 275
darkened printing ......................................... 274
database, terminfo ....................................... 237
Datamedia (example) .................................. 259
db ............................................................... 258
dch ............................................................. 259
dch1 ............................................................. 259
defc ............................................................ 275
definition, sharing ...................................... 268
def_prog_mode() .......................................... 66
def_shell_mode() .......................................... 66
delay .......................................................... 252, 263
delay_output() .............................................. 67
delch() ........................................................ 68
delete
effect on straddling character ...................... 21
delete/insert character ............................... 259
delete/insert line ......................................... 258
deletenl() ..................................................... 71
deletion ....................................................... 20
delscreen() .................................................. 72
delwin() ....................................................... 73
del_curterm() .............................................. 69
depth of input queue .................................. 204
derwin() ....................................................... 74
description of device .................................. 238
destructive scrolling .................................. 238
destructive tab ............................................ 207
device capability ......................................... 237
device name ................................................ 279
dialup terminal .......................................... 266
Digital LA100, LN03 .................................. 277
dim ............................................................. 260
direct cursor addressing ............................. 256
dl .............................................................. 258
dl1 ............................................................. 258
docr ............................................................ 273
dot-matrix graphics ..................................... 276
doupdate() ................................................... 75
draft-quality ............................................... 278
drawing a box .............................................. 255
dsl ............................................................. 263
dupwin() ...................................................... 76
EC ............................................................ 5, 8-9, 11-12, 16-17, 19-21, 23-24, 29, 237, 281
in <curses.h> .............................................. 220-225, 227-234
in <term.h> .................................................. 235
in addchstr() .............................................. 33
in addnstr() ................................................ 34
in addwstr() .............................................. 35
in add_wch() ............................................... 36
in add_wchnstr() ........................................ 37
in attr_get() ............................................... 39
in bkgd() .................................................... 43
in bkgrnd() ................................................ 44
in border() .................................................. 45
in border_set() .......................................... 46
in box_set() ............................................... 48
in can_change_color() ................................. 49
in chgat() ................................................... 53
in color_content() ........................................ 59
in COLOR_PAIRS ........................................... 60
in COLS ..................................................... 61
in copywin() .............................................. 62
in curscr .................................................... 63
in curs_set() .............................................. 64
in cur_term ................................................ 65
in delscreen() ............................................. 72
in del_curterm() ......................................... 69
in derwin() ................................................ 74
in dupwin() ............................................... 76
in echochar() ............................................. 78
in echo_wchar() .......................................... 79
in erasechar() ............................................ 82
Index

in filter() .................................................................83
in getbegyx() ..........................................................86
in getbkgd() ...............................................................88
in getbgrnd() ............................................................89
in getchar() ..............................................................90
in getmaxyx() ............................................................93
in getnstr() .............................................................94
in getn_wstr() ..........................................................96
in getparyx() ...........................................................98
in getwin() .............................................................101
in get_wch() ............................................................100
in get_wstr() ...........................................................102
in halfdelay() ..........................................................104
in has_colors() .........................................................105
in hline() ...............................................................107
in hline_set() ........................................................108
in idcok() ...............................................................109
in immedok() ........................................................111
in inchstr() ...........................................................113
in init_color() ........................................................114
in innstr() ............................................................117
in innwstr() ...........................................................118
in insdelw() ...........................................................120
in insdeln() ...........................................................122
in insnstr() ...........................................................124
in instr() ..............................................................125
in ins_nwstr() ........................................................123
in ins_wch() ...........................................................126
in ins_wstr() ...........................................................127
in instw() .............................................................131
in in_wch() ............................................................129
in in_wchnstr() .......................................................130
in isendwin() ........................................................132
in is_linetouched() ................................................133
in keyname() ........................................................134
in killchar() ..........................................................136
in LINES ...............................................................138
in meta() ..............................................................140
in mvcur() .............................................................144
in mvderwin() ........................................................145
in napms() ............................................................149
in newpad() ...........................................................150
in nqiflush() ........................................................157
in notimeout() .......................................................158
in pair_content() ...................................................160
in pechochar() ........................................................161
in putp() ...............................................................164
in putw() ..............................................................165
in qiflush() ...........................................................166
in redrawwin() .......................................................168
in restarting() .......................................................172
in ripoffline() .......................................................173
in scr() .................................................................177
in scr_char() ........................................................179
in setupterm() .......................................................183
in set_curt() ........................................................180
in slk_atroff() .......................................................184
in start_color() ......................................................187
in stdscr .............................................................188
in subpad() ...........................................................189
in synr() ...............................................................191
in termattr() ........................................................192
in termname() .........................................................193
in tgetent() ..........................................................194
in tigetflag() .........................................................196
in timeout() ..........................................................198
in touchline() ........................................................199
in tparm() .............................................................200
in tputs() ............................................................201
in ungetch() ........................................................204
in untouchwin() .....................................................205
in use_env() ..........................................................206
in vidattr() ...........................................................207
in vline() .............................................................209
in vline_set() ........................................................210
in vwprintw() ........................................................211
in vwscanw() ........................................................213
in vw_printw() .......................................................212
in vn_wchar() ........................................................214
in wunctrl() ........................................................218

ech .................................................................259
echo() ...............................................................267
echochar() ...........................................................78
echo_wchar() ........................................................79
ed .................................................................258
eighth bit .........................................................267
e1 .................................................................258
empty wide-character string ................................281
emulator, terminal .................................................265
en .................................................................257
enacs .............................................................260
end-of-line
  truncation/wrapping ...........................................19
eendwin() ...........................................................80
enhancement, turn off .........................................260
enter_ca_mode
  in tigetflag() ...................................................196
eo .................................................................260
erase character ................................................281
erase to end-of-line ..........................................258
Index

erase() .................................................................54, 81
erasechar() ..........................................................82
erasewchar() .........................................................82
error numbers .........................................................11
escape in terminfo ..................................................252
escape sequence .....................................................14
eslok() .................................................................263
estimating printer throughput .....................................278
et .................................................................257, 278
exit_ca_mode
  in tigetflag() .........................................................196
extra line of screen ................................................263
extra-bright character .............................................260
ff .................................................................266, 267
  delays ..............................................................263
filter() .................................................................83
first line in terminfo ..............................................238
flag, touched .........................................................14
flash .................................................................260
  delays ..............................................................252
flash() ..................................................................84
flashing screen .......................................................260
flow control ............................................................267
flushinp() ............................................................85
foreground colour ..................................................265
form feed ..............................................................273
format of entries ...................................................6
format of terminfo ..................................................238
fsl ......................................................................263
full duplex ............................................................77
function naming .....................................................26
generic terminal description ....................................266
getbegyx() ...........................................................86
getbkgd() .............................................................43, 88
getbkgrnd() ..........................................................44, 89
getchar() ...............................................................90
getch() ..................................................................91
getmaxyx() ...........................................................86, 93
getnstr() ...............................................................94
getn_wstr() ...........................................................96
getparyx() ...........................................................86, 98
getstr() ...............................................................94, 99
getwin() ...............................................................101
getyx() ...............................................................86, 103
get_wstrch() ........................................................100
get_wstr() ...........................................................96, 102
glitch, magic cookie .................................................260
glyph .................................................................20
gn .................................................................266, 267
grammar .............................................................239
graphic rendition, setting .........................................261
graphics, dot-matrix ...............................................276
  graphics, line-drawing..........................................264
  half line cursor movement .....................................266
  half-bright character ...........................................260
  halfdelay() .........................................................104
  has_colors() ......................................................49, 105
  has_ic() .............................................................106
  has_il() .............................................................106
  Hazeltine ..........................................................267
hc ......................................................................255
hd ......................................................................266
header line in terminfo ..........................................238
headers ...................................................................219
Heathkit H19 (example) ...........................................263
Hewlett-Packard
  model of colour specification ..................................265
  Hewlett-Packard 2621
    keypad ..................................................................262
    magic cookie glitch .............................................260
  Hewlett-Packard 2645 (example) .........................257
  high-order bit, setting ..........................................267
  highlighting .........................................................260
  hline() ..................................................................107
  hline_set() ..........................................................108
  hls ......................................................................265
  home .................................................................257
  hpa ..................................................................257
  hs ......................................................................263
  ht ......................................................................262
  hts ....................................................................262
  hu ......................................................................266
  hz ......................................................................267
  ich .......................................................................259
  ich1 .................................................................259
  idcok() ................................................................109
  idlok() ..............................................................55, 110
  if .........................................................................262
  il .......................................................................238
  il1 .....................................................................258
  immedok() ..........................................................111
  implementation-dependent .....................................4
  inch() ..................................................................112
  inchnstr() ...........................................................113
  inchstr() .............................................................113
  ind ......................................................................258
  delays ...............................................................255, 258
  independence of print modes assumed ......................274
  indn .................................................................255, 258
  initc .................................................................265
  initialisation .........................................................29, 262
  initialisation string .................................................29
Index

II ................................................................. 257
lm ............................................................. 267
locale ...................................................... 1
locale-specific ........................................ 16
long name of device ................................ 238
longname() ............................................. 139
lp1 .......................................................... 277
  recalculate resolution after .................... 270
lpix .......................................................... 277
lpix[x] ..................................................... 270
LSI ADM-3a (example) ............................ 257
lw ............................................................ 262
maddr ..................................................... 271
magic cookie glitch ................................ 260
mandatory delay ....................................... 252
manipulation of window ........................... 14
manual pages .......................................... 31
  format .................................................. 6
margin ..................................................... 255, 273
may ........................................................ 4
mc0, mc4, and so on ................................. 267
mcsex ....................................................... 269
mcub[1] ..................................................... 271
mcud[1] ..................................................... 271
mcuf[1] ..................................................... 271
mcuu[1] ..................................................... 271
mcu[b/d/f/u][1] .......................................... 272
media copy string .................................... 267
meta key ................................................. 140, 267
meta() ..................................................... 140
mgc ........................................................ 263, 274
mhpa ....................................................... 271
  reverse motion should not affect .............. 272
Micro-Term ACT-IV (example) ................. 257
Micro-Term MIME (example) .................... 260
mir ........................................................ 259
mjum ....................................................... 271
mls ......................................................... 269
modification outside subwindow ............... 21
motion, automatic .................................... 274
move() ...................................................... 141
mrcup ..................................................... 256
msgr ......................................................... 260
  enhanced printing .................................. 274
multi-byte character
  function naming .................................. 26
multi-column character ......................... 16
multiple character functions ................. 26
must ....................................................... 4
mv ........................................................ 142
mv prefix .............................................. 26
mv prefix .............................................. 26
position arguments ................................ 19
mvaddch() ........................................... 32
mvaddchnstr() ..................................... 33
mvaddchstr() ....................................... 33
mvaddnstr() ......................................... 34
mvaddnwstr() ....................................... 35
mvaddstr() .......................................... 34
mvaddwstr() ......................................... 35
mvadd_wch() ......................................... 36
mvadd_wchnstr() .................................. 37
mvadd_wchstr() .................................... 37
mvchgat() ........................................... 53
mvcur() ................................................ 144
mvdelch() ............................................ 68
mvderwin() .......................................... 145
mvgetch() ............................................ 91
mvgetnstr() .......................................... 94
mvgetn_wstr() ....................................... 96
mvgetstr() ............................................ 94
mvgetw_str() ......................................... 100
mvget_wstr() ......................................... 96
mvhline() ............................................. 107
mvhline_set() ....................................... 108
mvinsch() ........................................... 119
mvinsnstr() .......................................... 122
mvinsstr() ........................................... 122
mvprintw() ........................................... 146
mvprintn() ........................................... 147
mvsanw() ............................................. 107
mvsanw() ............................................. 107
mvw_prefix .......................................... 26
mvwaddch() ......................................... 32
mvwaddchnstr() .................................. 33
mvwaddchstr() ..................................... 33
mvwaddnstr() ....................................... 34
mvwaddnwstr() .................................... 35
mxprefix ................................................. 26

<table>
<thead>
<tr>
<th>term</th>
<th>page number(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>page eject</td>
<td>266</td>
</tr>
<tr>
<td>pairs</td>
<td>265</td>
</tr>
<tr>
<td>pair_content()</td>
<td>49, 160</td>
</tr>
<tr>
<td>PAIR_NUMBER()</td>
<td>49, 160</td>
</tr>
<tr>
<td>parametrised string</td>
<td>256</td>
</tr>
<tr>
<td>parent window</td>
<td>14, 281</td>
</tr>
<tr>
<td>patch</td>
<td>266</td>
</tr>
<tr>
<td>pb</td>
<td>263</td>
</tr>
<tr>
<td>PC terminal emulator</td>
<td>265</td>
</tr>
<tr>
<td>pechochar()</td>
<td>161</td>
</tr>
<tr>
<td>pecho_wchar()</td>
<td>161</td>
</tr>
<tr>
<td>period in terminfo</td>
<td>253</td>
</tr>
<tr>
<td>Perkin-Elmer Owl (example)</td>
<td>259</td>
</tr>
<tr>
<td>pfkey</td>
<td>262</td>
</tr>
<tr>
<td>pfloc</td>
<td>262</td>
</tr>
<tr>
<td>pfz</td>
<td>262</td>
</tr>
<tr>
<td>pln</td>
<td>262</td>
</tr>
<tr>
<td>pnoutrefresh()</td>
<td>150, 162</td>
</tr>
<tr>
<td>pop-up window</td>
<td>21</td>
</tr>
<tr>
<td>porder</td>
<td>276</td>
</tr>
<tr>
<td>position</td>
<td>26</td>
</tr>
<tr>
<td>current or specified</td>
<td>26</td>
</tr>
<tr>
<td>postfix</td>
<td>256</td>
</tr>
<tr>
<td>prefix on function/argument</td>
<td>26</td>
</tr>
<tr>
<td>prefresh()</td>
<td>150</td>
</tr>
<tr>
<td>print quality</td>
<td>278</td>
</tr>
<tr>
<td>printer resolution</td>
<td>268</td>
</tr>
<tr>
<td>printer specification in terminfo</td>
<td>268</td>
</tr>
<tr>
<td>printing rate</td>
<td>278</td>
</tr>
<tr>
<td>printw()</td>
<td>146, 163</td>
</tr>
<tr>
<td>property</td>
<td></td>
</tr>
<tr>
<td>background</td>
<td>18</td>
</tr>
<tr>
<td>rendition</td>
<td>18</td>
</tr>
<tr>
<td>window</td>
<td>18</td>
</tr>
<tr>
<td>proportional delay</td>
<td>252</td>
</tr>
<tr>
<td>proportional printing</td>
<td>268</td>
</tr>
<tr>
<td>prot</td>
<td>260</td>
</tr>
<tr>
<td>protected text</td>
<td>260</td>
</tr>
<tr>
<td>protocol (xon/xoff)</td>
<td>255</td>
</tr>
<tr>
<td>putp()</td>
<td>164</td>
</tr>
<tr>
<td>putwin()</td>
<td>101, 165</td>
</tr>
<tr>
<td>qiflush()</td>
<td>157, 166</td>
</tr>
<tr>
<td>quality of printing</td>
<td>278</td>
</tr>
<tr>
<td>raster graphics</td>
<td>276</td>
</tr>
<tr>
<td>raw()</td>
<td>52, 167</td>
</tr>
<tr>
<td>rbim</td>
<td>276</td>
</tr>
<tr>
<td>rc</td>
<td>258, 263</td>
</tr>
<tr>
<td>inclusion in tsl/fsl</td>
<td>263</td>
</tr>
<tr>
<td>rcsd</td>
<td>275</td>
</tr>
<tr>
<td>reading subwindow</td>
<td></td>
</tr>
<tr>
<td>effect on straddling character</td>
<td>21</td>
</tr>
<tr>
<td>redrawwin()</td>
<td>168</td>
</tr>
<tr>
<td>refresh</td>
<td>14</td>
</tr>
<tr>
<td>clears touched flag</td>
<td>14</td>
</tr>
<tr>
<td>refresh()</td>
<td>75, 169</td>
</tr>
<tr>
<td>relocation of cursor</td>
<td>20</td>
</tr>
<tr>
<td>rendition</td>
<td>17, 261, 281</td>
</tr>
<tr>
<td>background</td>
<td>18</td>
</tr>
<tr>
<td>window</td>
<td>18</td>
</tr>
<tr>
<td>rendition of character placed in window</td>
<td>22</td>
</tr>
<tr>
<td>rep</td>
<td>266</td>
</tr>
<tr>
<td>replacing characters</td>
<td>19</td>
</tr>
<tr>
<td>resetty()</td>
<td>171</td>
</tr>
<tr>
<td>reset_prog_mode()</td>
<td>66, 170</td>
</tr>
<tr>
<td>reset_shell_mode()</td>
<td>66</td>
</tr>
<tr>
<td>resolution</td>
<td>268</td>
</tr>
<tr>
<td>resolution, effect of changing</td>
<td>277</td>
</tr>
<tr>
<td>restatterm()</td>
<td>69, 172</td>
</tr>
<tr>
<td>restoring subwindow</td>
<td>21</td>
</tr>
<tr>
<td>rev</td>
<td>260</td>
</tr>
<tr>
<td>reverse Polish</td>
<td>256</td>
</tr>
<tr>
<td>reverse-video screen</td>
<td>260</td>
</tr>
<tr>
<td>rf</td>
<td>263</td>
</tr>
<tr>
<td>rfi</td>
<td>266</td>
</tr>
<tr>
<td>ri</td>
<td>255, 258</td>
</tr>
<tr>
<td>right margin</td>
<td>273</td>
</tr>
<tr>
<td>right-to-left writing</td>
<td>2</td>
</tr>
<tr>
<td>rin</td>
<td>255, 258</td>
</tr>
<tr>
<td>ripoffline()</td>
<td>173</td>
</tr>
<tr>
<td>ritm</td>
<td>274</td>
</tr>
<tr>
<td>rlm</td>
<td>272</td>
</tr>
<tr>
<td>rmacs</td>
<td>260</td>
</tr>
<tr>
<td>rmcup</td>
<td>258</td>
</tr>
<tr>
<td>in scr_dump()</td>
<td>176</td>
</tr>
<tr>
<td>rmdc</td>
<td>259</td>
</tr>
<tr>
<td>rmicm</td>
<td>272</td>
</tr>
<tr>
<td>rmir</td>
<td>259</td>
</tr>
<tr>
<td>rmkx</td>
<td>262</td>
</tr>
<tr>
<td>rmmin</td>
<td>262</td>
</tr>
<tr>
<td>rmm</td>
<td>267</td>
</tr>
<tr>
<td>rmp</td>
<td>259</td>
</tr>
<tr>
<td>rmso</td>
<td>260</td>
</tr>
<tr>
<td>rmxm</td>
<td>267</td>
</tr>
<tr>
<td>rounding</td>
<td>268</td>
</tr>
<tr>
<td>row or column cursor addressing</td>
<td>257</td>
</tr>
<tr>
<td>RPN</td>
<td>256</td>
</tr>
<tr>
<td>rs1, rs2</td>
<td>263</td>
</tr>
<tr>
<td>rshm</td>
<td>274</td>
</tr>
<tr>
<td>rsrshm</td>
<td>274</td>
</tr>
<tr>
<td>rsrshm</td>
<td>274</td>
</tr>
<tr>
<td>rsupm</td>
<td>224</td>
</tr>
<tr>
<td>rum</td>
<td>272</td>
</tr>
</tbody>
</table>
Index

rwidm.................................................................274
sam.................................................................272
savetty( ).........................................................171, 174
sbim .................................................................276
sc .................................................................258, 263
  inclusion in tsl/fsl .......................................263
scanw( ).........................................................147, 175
scp .................................................................265
screen ............................................................14, 255
SCREEN.............................................................281
screen blink .....................................................260
scrl( ) .................................................................177
scroll
  effect on straddling character .....................21
scroll( )............................................................177
scrolling ...........................................................255
scrolling region ...............................................258
scrollok()..........................................................55, 178
scr_dump()........................................................275
scr_init()............................................................176
scr_restore().....................................................176
scr_set() ............................................................176
scls .................................................................275
scsd .................................................................275
sdrfq .................................................................278
search path for TERM ........................................279
setab ...............................................................265
setaf ...............................................................265
setb .................................................................265
setcchar( )..........................................................179
setf.................................................................265
setscregs().......................................................55, 181
settabterm().....................................................258
setupterm()........................................................69, 183
set_curterm().....................................................69, 180
set_term()............................................................182
sgr .................................................................261
sgro .................................................................260
shadow .............................................................274
shadowing .........................................................274
sharing definition in terminfo .........................268
should...............................................................4
signals, relationship to ....................................13
similar terminal ...............................................268
single-byte character
  function naming ...............................................26
sitm ...............................................................274
slash
  in terminfo .....................................................252
slk_attroff() ....................................................184
slk_attron()......................................................184
slk_attribute..................................................184
slk_attr_set()....................................................184
slk_clear()........................................................184
slk_color().......................................................184
slk_init()..........................................................184
slk_label()........................................................184
slk_noutrefresh()...............................................184
slk_refresh().....................................................184
slk_restore().....................................................184
slk_set()............................................................184
slk_touch()........................................................184
slk_wset()........................................................184
slm .................................................................272
smacs ..............................................................260
smb[l/r/t] .........................................................273
smcup .............................................................258
smdc ...............................................................259
smg[l/r/t]p .........................................................273
smicm ..............................................................272
smir .................................................................259
smkx ...............................................................262
smln ...............................................................262
smm .................................................................267
smso .................................................................260
smul .................................................................260
smxon ..............................................................267
snlq .................................................................278
snmq ...............................................................278
space
  use in terminfo ...............................................238
space character
  resulting rendition .........................................22
spacing complex character ...............................17
spacing of characters .......................................268
special keys .....................................................262
special mode ....................................................258
special mode of device .....................................279
speed of printing .............................................278
spinh ..............................................................276
spinv ..............................................................276
sshm ..............................................................274
ssubm ..............................................................274
ssupm ..............................................................274
stack in terminfo .............................................256
standend().......................................................186
standout().......................................................186
standout mode ................................................260
start_color()....................................................49, 187
status line .......................................................263
stdscr .................................................................14, 188
current character available .................................274
current character straddling border ......................21
current string capability ......................................238
clearing character ..............................................262
clearing expansion .............................................257
clearing special processing .................................22
clearing use in terminfo .....................................238
clearing tab stop ................................................29
clearing tab character ........................................262
Tektronix model of colour specification .................265
Tektronix command character ...............................266
Tektronix 4025 (example) ...................................257
Telerik destructive tab .......................................267
Telerik 1061 (example) .......................................260
TERM in initscr( ) ..............................................115
termmats( ) .......................................................192
terminal ..........................................................15
terminal emulator .............................................265
terminal-independence ......................................1, 237
terminfo ..........................................................237
TERMINFO ..........................................................279
terminfo format ...............................................238
terminology .......................................................4
termmname( ) .....................................................193	getent( ) ..........................................................194
tigetflag( ) .........................................................194
throughput ......................................................278
tigetnum( ) .........................................................196
tigetstr( ) ..........................................................196
tilde, inability to display ....................................267
timeout( ) ..........................................................158, 198
top and left edge ..............................................255
touch ..............................................................281
touched ................................................................14
touchline( ) ........................................................133, 199
touchwin( ) .......................................................133
tparm( ) ..............................................................196, 200
tputs( ) ...............................................................164, 201
truncation ..........................................................19
tsl ........................................................................263
TVI 9122 (example) ...........................................260

typeahead discarding ...........................................85
typeahead( ) .......................................................202
uc ........................................................................260
ul ........................................................................260
unctrl( ) ..............................................................203
undefined ............................................................4
underline cursor ..................................................260
underlining .......................................................260
ungetch( ) ...........................................................204
unget_wch( ) .......................................................204
uniqueness of terminfo aliases ............................238
unspecified .......................................................14
untouchwin( ) .....................................................133, 205
update sets touched flag ....................................14
user use .............................................................268
user preference for use of device .........................279
use_env( ) ...........................................................206
variability in print rate .......................................278
variable-width font .........................................268
vertical bar use in terminfo ...............................238
vi use of terminfo ..............................................237
vidattr( ) ...........................................................207
video attribute ..................................................16
video enhancement, turn off ..............................260
vidputs( ) ...........................................................207
vid_attr() ...........................................................207
vid_puts() ...........................................................207
virtual terminal ................................................266
visible bell .......................................................260