## Contents

### Chapter 1 Introduction

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>This Document</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1</td>
<td>Relationship to Previous Issues</td>
<td>1</td>
</tr>
<tr>
<td>1.1.2</td>
<td>Features Introduced in Issue 7</td>
<td>2</td>
</tr>
<tr>
<td>1.1.3</td>
<td>Features Withdrawn in Issue 7</td>
<td>2</td>
</tr>
<tr>
<td>1.1.4</td>
<td>Features Introduced in Issue 4</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Conformance</td>
<td>3</td>
</tr>
<tr>
<td>1.2.1</td>
<td>Base Curses Conformance</td>
<td>3</td>
</tr>
<tr>
<td>1.2.2</td>
<td>Enhanced Curses Conformance</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>Terminology</td>
<td>4</td>
</tr>
<tr>
<td>1.3.1</td>
<td>Shaded Text</td>
<td>5</td>
</tr>
<tr>
<td>1.4</td>
<td>Format of Entries</td>
<td>6</td>
</tr>
</tbody>
</table>

### Chapter 2 General Information

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Use and Implementation of Interfaces</td>
<td>9</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Use and Implementation of Functions</td>
<td>9</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Use and Implementation of Macros</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>The Compilation Environment</td>
<td>10</td>
</tr>
<tr>
<td>2.2.1</td>
<td>The X/Open Name Space (ENHANCED CURSES)</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>Data Types</td>
<td>12</td>
</tr>
</tbody>
</table>

### Chapter 3 Interface Overview

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Components</td>
<td>13</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Relationship to the XSH Specification</td>
<td>13</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Relationship to the XBD Specification</td>
<td>13</td>
</tr>
<tr>
<td>3.2</td>
<td>Screens, Windows, and Terminals</td>
<td>14</td>
</tr>
<tr>
<td>3.3</td>
<td>Characters</td>
<td>15</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Character Storage Size</td>
<td>15</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Multi-Column Characters</td>
<td>16</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Attributes</td>
<td>16</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Rendition</td>
<td>16</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Non-Spacing Characters</td>
<td>16</td>
</tr>
<tr>
<td>3.3.6</td>
<td>Window Properties</td>
<td>17</td>
</tr>
<tr>
<td>3.4</td>
<td>Conceptual Operations</td>
<td>18</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Screen Addressing</td>
<td>18</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Basic Character Operations</td>
<td>18</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Special Characters</td>
<td>20</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Rendition of Characters Placed into a Window</td>
<td>21</td>
</tr>
<tr>
<td>3.5</td>
<td>Input Processing</td>
<td>22</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Keypad Processing</td>
<td>22</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Input Mode</td>
<td>23</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Delay Mode</td>
<td>24</td>
</tr>
<tr>
<td>3.5.4</td>
<td>Echo Processing</td>
<td>24</td>
</tr>
<tr>
<td>3.6</td>
<td>The Set of Curses Functions</td>
<td>24</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Function Name Conventions</td>
<td>24</td>
</tr>
</tbody>
</table>
3.6.2 Function Families Provided ........................................... 25
3.7 Interfaces Implemented as Macros .................................... 26
3.8 Initialized Curses Environment ........................................ 27
3.9 Synchronous and Networked Asynchronous Terminals ......... 27

Chapter 4 Curses Interfaces .................................................. 29

Chapter 5 Headers .................................................................. 305

Chapter 6 Utilities .................................................................. 323

Chapter 7 Terminfo Source Format (ENHANCED CURSES) .... 337

Appendix A Application Usage .................................................. 353
Preface

The Open Group

The Open Group is a vendor-neutral and technology-neutral consortium, whose vision of Boundaryless Information Flow™ will enable access to integrated information within and between enterprises based on open standards and global interoperability. The Open Group works with customers, suppliers, consortia, and other standards bodies. Its role is to capture, understand, and address current and emerging requirements, establish policies, and share best practices; to facilitate interoperability, develop consensus, and evolve and integrate specifications and Open Source technologies; to offer a comprehensive set of services to enhance the operational efficiency of consortia; and to operate the industry’s premier certification service, including TOGAF™ and UNIX® certification.

Further information on The Open Group is available at www.opengroup.org.

The Open Group has over 20 years’ experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification.

More information is available at www.opengroup.org/certification.

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical and Product Standards and Guides, but which also includes white papers, technical studies, branding and testing documentation, and business titles. Full details and a catalog are available at www.opengroup.org/bookstore.

As with all live documents, Technical Standards and Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards-compatible and those which are not:

- A new Version indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it replaces the previous publication.

- A new Issue indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published at www.opengroup.org/corrigenda.

This Document

This Technical Standard 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 Technical Standard is structured as follows:

- Chapter 1 introduces Curses, gives an overview of enhancements that have been made to this version, and lists specific interfaces that have been withdrawn. This chapter also defines the requirements for conformance to this document and shows the generic format followed by interface definitions in Chapter 4.
Preface

- **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 the functions and global variables, and define types, constants, macros, and data structures that are needed by programs using the services provided by **Chapter 4**.

- **Chapter 6** replaces the specification of the `tput` utility in the XCU specification and defines additional Curses utilities.

- **Chapter 7** 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** 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.
Preface

- 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 pages may provide additional or more specific portability warnings about use of the material.

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

The material on pages labeled 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>.
Trademarks

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

Boundaryless Information Flow™ and TOGAF™ are trademarks and Motif®, Making Standards Work®, OSF/1®, The Open Group®, UNIX®, and the “X” device are registered trademarks of The Open Group in the United States and other countries.

Hewlett-Packard®, HP®, HP-UX®, and Openview® are registered trademarks of Hewlett-Packard Company.

The names of terminals and of terminal manufacturers cited as examples in Chapter 7 and Appendix A may be trademarks, which are the property of their respective owners.
Acknowledgements

The Open Group 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.
The following documents are referenced in this Technical Standard:

**ANSI C**

**ISO/IEC 646**

**ISO/IEC 6429: 1992**
Information Technology — Control Functions for Coded Character Sets.

**ISO/IEC 10646**

**ISO 2022**

**ISO 8859-1**

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

**SVID, Issue 2**

**SVID, Issue 3**

**System V Release 2.0**

**System V Release 4.2**

The following documents published by The Open Group are referenced in this Technical Standard:

**Base Specifications, Issue 5**
Technical Standard, February 1997, published by The Open Group:

Referenced Documents


Base Specifications, Issue 6
Technical Standard, April 2004, published by The Open Group:

Base Specifications, Issue 7

Issue 2

Issue 3

Issue 4

Issue 4, Version 2

Issue 7
This standard.
Chapter 1

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 recognize the codeset 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 27) are considered.

1.1 This Document

This document is Issue 7.

1.1.1 Relationship to Previous Issues

Relationship to Issue 3

The unshaded material in this document preserves syntactic compatibility with the Curses specification, Issue 3, except that some functions have been withdrawn (see Section 1.1.3, on page 2). In addition, retained interfaces from the Curses specification, Issue 3 have been clarified as a result of industry feedback.

Relationship between Issue 4, Version 1 and Issue 4, Version 2

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

Relationship between Issue 4, Version 2 and Issue 7

Issue 7 is updated as follows:

- Functionality marked “To Be Withdrawn” is removed.
- Clarification is added to explain that the int arguments passed to getbegyx(), getmaxyx(), getparyx(), and getyx() must be modifiable lvalues.
- The tparm() function is marked obsolescent.
- Features described in Section 1.1.2 (on page 2) are introduced.
1.1.2 Features Introduced in Issue 7

The following features are introduced in Issue 7:

- Function prototypes are updated to use `const` where appropriate.
- The `tiparm()` function is added.
- The following new utilities are added in Chapter 6:

  - `infocmp`
  - `tic`
  - `tput`
  - `untic`

1.1.3 Features Withdrawn in Issue 7

The following interfaces are withdrawn in this document:

<table>
<thead>
<tr>
<th>Withdrawn Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tgetent()</code></td>
</tr>
<tr>
<td><code>tgetflag()</code></td>
</tr>
<tr>
<td><code>tgetnum()</code></td>
</tr>
<tr>
<td><code>tgoto()</code></td>
</tr>
<tr>
<td><code>vwprintw()</code></td>
</tr>
<tr>
<td><code>vwscanw()</code></td>
</tr>
</tbody>
</table>

1.1.4 Features Introduced in Issue 4

The following features were introduced in Issue 4.

**Internationalization**

This version of the Curses specification has been enhanced to support a wide range of internationalized 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 codesets. The actual codesets 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 codesets (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 8859-1:1987. Many traditional implementations only supported ISO/IEC 646:1991 and preceding codeset specifications, in which the length of a character was an octet.

The primary standardization issue with the increasing size of a character is that neither the ANS X3.159-1989 or 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, The Open Group 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 generalized to permit both right-to-left and left-to-right writing. This document does not specify whether the implementation supports more than one direction of writing. The behavior of the interfaces in this document 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 behavior of the chtype interfaces in this document 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 generalized 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 color terminals, printers, modems, and mice.

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, on page 4).

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 document except that it need not support those occurring on reference pages labeled ENHANCED CURSES and in shaded areas marked with the EC margin legend.
- The chtype data type shall support at least octet-based codesets, such as ISO 8859-1: 1987.
- The system may provide additional or enhanced interfaces, headers, and facilities not required by this document, provided that such additions or enhancements do not affect the behavior 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 document occurring on reference pages labeled ENHANCED CURSES and in shaded areas marked with the EC margin legend.
• The system may provide additional or enhanced interfaces, headers, and facilities not required by this document, provided that such additions or enhancements do not affect the behavior of an application that requires only the facilities described in this document.

1.3 Terminology

The following terms are used in this document:

can
Describes a permissible optional feature or behavior available to the user or application. The feature or behavior is mandatory for an implementation that conforms to this document. An application can rely on the existence of the feature or behavior.

implementation-defined
Describes a value or behavior that is not defined by this document but is selected by an implementor. The value or behavior may vary among implementations that conform to this document. An application should not rely on the existence of the value or behavior. An application that relies on such a value or behavior cannot be assured to be portable across conforming implementations.

The implementor shall document such a value or behavior so that it can be used correctly by an application.

legacy
Describes a feature or behavior that is being retained for compatibility with older applications, but which has limitations which make it inappropriate for developing portable applications. New applications should use alternative means of obtaining equivalent functionality.

may
Describes a feature or behavior that is optional for an implementation that conforms to this document. An application should not rely on the existence of the feature or behavior. An application that relies on such a feature or behavior cannot be assured to be portable across conforming implementations.

To avoid ambiguity, the opposite of may is expressed as need not, instead of may not.

must
Describes a feature or behavior that is mandatory for an application or user. An implementation that conforms to this document shall support this feature or behavior.

shall
Describes a feature or behavior that is mandatory for an implementation that conforms to this document. An application can rely on the existence of the feature or behavior.

should
For an implementation that conforms to this document, describes a feature or behavior that is recommended but not mandatory. An application should not rely on the existence of the
feature or behavior. An application that relies on such a feature or behavior cannot be assured to be portable across conforming implementations.

For an application, describes a feature or behavior that is recommended programming practice for optimum portability.

**undefined**
Describes the nature of a value or behavior not defined by this document which results from use of an invalid program construct or invalid data input.

The value or behavior may vary among implementations that conform to this document. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.

**unspecified**
Describes the nature of a value or behavior not specified by this document which results from use of a valid program construct or valid data input.

The value or behavior may vary among implementations that conform to this document. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.

**will**
Same meaning as *shall*; *shall* is the preferred term.

### 1.3.1 Shaded Text

Shaded text in this document is qualified by a code in the left margin. The codes and their meanings are as follows:

**EC**
X/Open Enhanced Curses
The functionality described 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 pages may provide additional or more specific portability warnings about use of the material.

If an entire SYNOPSIS section is shaded and marked EC, all the functionality described on that reference page is an extension.

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

**OB**
Obsolescent
The functionality described may be removed in a future version of this document. Applications should not use obsolescent features.

Where applicable, the material is identified by use of the OB margin legend.
1.4 **Format of Entries**

The entries in Chapter 4 and Chapter 5 are based on a common format, as follows. The only sections relating to conformance are the SYNOPSIS, DESCRIPTION, RETURN VALUE, and ERRORS sections.

**NAME**

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

**SYNOPSIS**

This section summarizes 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:

```
#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 is 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 error values returned by a function or stored into a variable accessed through the symbol `errno` if an error occurs.

“No errors are defined” means that error values returned by a function or stored into a variable accessed through the symbol `errno`, if any, depend on the implementation.

**EXAMPLES**

This section is informative.

This section gives examples of usage, where appropriate.

**APPLICATION USAGE**

This section is informative.

This section gives warnings and advice to application developers about the entry.

**RATIONALE**

This section is informative.

This section contains historical information concerning the contents of the entry and why features were included or discarded by the developers of this document.

**FUTURE DIRECTIONS**

This section is informative.

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 is informative.
This section gives references to related information.

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

The entries in Chapter 6 are in the same format as the utility reference pages in the XCU specification (see the XCU specification, Section 1.4, Utility Description Defaults).
Chapter 2

General Information

2.1 Use and Implementation of Interfaces

2.1.1 Use and Implementation of Functions

Each of the following statements shall apply to all functions unless explicitly stated otherwise in the detailed descriptions that follow:

1. 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 behavior is undefined.

2. Any function declared in a header may also be implemented as a macro defined in the header, so a 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 function even if it is also defined as a macro. The use of the C-language `#undef` construct to remove any such macro definition shall also ensure that an actual function is referred to.

3. Any invocation of a function that is implemented as a macro shall 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.

4. Provided that a function can be declared without reference to any type defined in a header, it is also permissible to declare the function explicitly and use it without including its associated header.

5. If a function that accepts a variable number of arguments is not declared (explicitly or by including its associated header), the behavior is undefined.

2.1.2 Use and Implementation of Macros

Each of the following statements shall apply to all macros unless explicitly stated otherwise:

1. Any definition of an object-like macro in a header shall expand to code that is fully protected by parentheses where necessary, so that it groups in an arbitrary expression as if it were a single identifier.

2. Any definition of a function-like macro in a header shall expand to code that evaluates each of its arguments exactly once, fully protected by parentheses where necessary, so that it is generally safe to use arbitrary expressions as arguments.
3. Any definition of a function-like macro in a header can be invoked in an expression anywhere a function with a compatible return type could be called.

2.2 The Compilation Environment

The compilation environment in this document can exist in the following environment:

- Base Specifications, Issue 7

The compilation environment is defined as follows:

Applications shall ensure that the feature test macro _XOPEN_SOURCE is defined with the value 700 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.

In the compilation of an application that #defines the _XOPEN_SOURCE feature test macro, no header defined by this document or by the Base Specifications, Issue 7 shall be included prior to the definition of the feature test macro. This restriction also applies to any implementation-defined header in which these feature test macros are used. If the definition of the macro does not precede the #include, the result is undefined.

Identifiers in this document may only be undefined using the #undef directive as described in Section 2.1.1 (on page 9) or Section 2.2.1. These #undef directives must follow all #include directives of any XSI headers.

Since this document is aligned with the ISO C Standard, and since all functionality enabled by _POSIX_C_SOURCE set equal to 200809L is enabled by _XOPEN_SOURCE set equal to 700, there should be no need to define _POSIX_C_SOURCE if _XOPEN_SOURCE is so defined. Therefore, if _XOPEN_SOURCE is set equal to 700 and _POSIX_C_SOURCE is set equal to 200809L, the behavior is the same as if only _XOPEN_SOURCE is defined and set equal to 700. However, should _POSIX_C_SOURCE be set to a value greater than 200809L, the behavior is unspecified.

The c99 utility shall recognize the following additional −I option for standard libraries:

−l curses This option shall make available all interfaces referenced in this document (except for those labeled ENHANCED CURSES and except for portions marked with the EC margin legend).

If the implementation defines _XOPEN_CURSES, then −l curses shall also make available all interfaces referenced in this document and labeled ENHANCED CURSES and portions marked with the EC margin legend.

It is unspecified whether the library libcurses.a 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.

Implementations may also add members to a structure or union without controlling the visibility of those members with a feature test macro, as long as a user-defined macro with the same name cannot interfere with the correct interpretation of the program.

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 two (2) uppercase 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 #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. With the exception of identifiers beginning with the prefix _POSIX_, all identifiers that begin with an underscore and either an uppercase 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.
4. 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.

2.3 Data Types

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

- **attr_t**
  An integer 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 <curses.h> that begin with the prefix WA_.

- **bool**
  As described in <stdbool.h>.

- **ctype**
  An integer 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 <curses.h> that have the A_ prefix. The application can extract these components of a ctype value using the base masks defined in <curses.h> for this purpose.

- **wchar_t**
  As described in <stddef.h>.

- **cchar_t**
  A type that can reference a string of wide characters of up to an implementation-defined length, a color-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 an empty wide-character string. Arrays of cchar_t objects are terminated by a null cchar_t object.

- **SCREEN**
  An opaque terminal representation.

- **wint_t**
  As described in <wchar.h>.

- **wchar_t**
  As described in <stddef.h>.

- **WINDOW**
  An opaque window representation.
3.1 Components

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

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

3.1.1 Relationship to the XSH Specification

\textbf{Error Numbers}

Most functions provide an error number in \texttt{errno}, which is a symbol defined or declared in \texttt{<errno.h>} as either a macro or an identifier declared with external linkage; the symbol expands to a modifiable lvalue of type \texttt{int}.

A list of valid values for \texttt{errno} and advice to application writers on the use of \texttt{errno} appears in the XSH specification.

\textbf{Signals}

Curses implementations may provide for special handling of the SIGINT, SIGQUIT, and SIGTSTP signals if their disposition is SIG_DFL at the time \texttt{initscr()} is called (see \texttt{initscr()}, on page 126).

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 \texttt{sigaction()} in the XSH specification).

\textbf{Thread-Safety}

The interfaces defined by this document need not be thread-safe.

3.1.2 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 11, General Terminal Interface.

There is no requirement that the paradigms that exist while in Curses mode be carried over outside the Curses environment (see \texttt{def_prog_mode()}).
Signals
The behavior of Curses with respect to signals not defined by the XBD 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.

Window
The Curses functions permit manipulation of window objects, which can be thought of as two-dimensional arrays of characters and their renditions representing all or part of a terminal’s physical screen. Windows do not have to correspond to the entire screen. It is possible to create smaller windows and also to indicate that a window is only partially visible on the screen. It is possible to create windows larger than the terminal screen using pads. A default window called stdscr, which is the size of the terminal screen, is supplied. Others may be created with newterm().

Data structures declared as WINDOW refer to windows (and to subwindows, derived windows, pads, and subpads, as described elsewhere). These data structures are manipulated with functions described in Chapter 7.

Among the most basic functions are move() and addch() which manipulate the default window stdscr, and refresh() which tells Curses to update the user’s screen from stdscr. More general versions of these functions enable specific windows to be manipulated and refreshed.

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 141.).)

Subwindow
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.
Ancestor
The term *ancestor* refers to a window’s parent, or its parent, and so on.

Derived Window
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.

Pad
A pad is a specialized case of a window which can be bigger than the actual screen size and is not necessarily associated with a particular part of the screen. Pads should be used whenever a window larger than the terminal screen is required.

Subpad
A subpad is a specialized case of a window created within another pad.

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, on page 16).
- Some characters may be non-spacing characters, defined only in association with a spacing character (see Section 3.3.5, on page 16).
- 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 color 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 color 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.

Colors are always used in pairs (referred to as color-pairs). A color-pair consists of a foreground color (for characters) and a background color (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 `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 five (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 initialized using `setcchar()` and its contents can be extracted using `getcchar()`. The behavior of functions that take a `cchar_t` input argument is undefined if the application provides a `cchar_t` value that was not initialized 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 21).
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 21)

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: 1987 character set (with left-to-right writing), \((0, 0)\) represents the upper left-hand corner of the screen.

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

3.4.2 Basic Character Operations

Adding (Overwriting)
The Curses functions that contain the word \(add\)—such as \(addch()\)—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 16).

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 behavior 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 scrollok() 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. Since none of the insertion functions support wrapping, the characters that would thus extend beyond the end of the line are removed from the window. This may produce orphaned columns.

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. Portable applications should use getyx() to obtain the current cursor position in a window.
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_uch()` 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.
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.

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.

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.

If `lines` equals one, the behavior is unspecified (note that the `filter()` function sets `lines` equal to one).

Tab characters in text move subsequent characters to the next horizontal tab stop. Curses may assume that 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 uppercase 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 color that the application specifies; or the window color, if the application does not specify a color
- The attributes specified, OR’ed with the window attributes

If the character is the `<space>` character, then the window receives:

- The background character
- The color that the application specifies; or the window color, if the application does not specify a color
• 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 360).

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 20) are defined or redefined to be characters that are members of a function key sequence, then Curses will be unable to recognize 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>
<tbody>
<tr>
<td>Cooked Mode</td>
<td>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.</td>
</tr>
<tr>
<td>cbreak Mode</td>
<td>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.</td>
</tr>
<tr>
<td>Half-Delay Mode</td>
<td>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.</td>
</tr>
<tr>
<td>Raw Mode</td>
<td>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.</td>
</tr>
</tbody>
</table>

The terminal interface settings are recorded when the process calls initscr() or newterm() to initialize 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 case the initial input mode is cbreak mode.

The behavior 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; color 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 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 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 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 reference 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]waddch()</td>
<td>Add (Overwrite)</td>
<td></td>
<td></td>
<td></td>
<td>addch()</td>
</tr>
<tr>
<td>[mv]waddch[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]wadd[n]str()</td>
<td>Add a string</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>addnstr()</td>
</tr>
<tr>
<td>[mv]wadd[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]wadd_wch()</td>
<td>Add a wide character and rendition</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>add_wch()</td>
</tr>
<tr>
<td>[mv]wchgat()</td>
<td>Change Renditions</td>
<td></td>
<td></td>
<td></td>
<td>chgat()</td>
</tr>
<tr>
<td>[mv]wdelch()</td>
<td>Delete</td>
<td></td>
<td></td>
<td></td>
<td>delch()</td>
</tr>
<tr>
<td>[mv]wgetch()</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]wget[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]wget_wch()</td>
<td>Get a wide character</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>get_wch()</td>
</tr>
<tr>
<td>[mv]wget[n]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>Explicit Cursor Movement</td>
<td></td>
<td></td>
<td></td>
<td>move()</td>
</tr>
<tr>
<td>[mv]winch()</td>
<td>Input (Read Back from Window)</td>
<td></td>
<td></td>
<td></td>
<td>inch()</td>
</tr>
<tr>
<td>[mv]winch[n]str()</td>
<td>Input an array of characters and attributes</td>
<td></td>
<td></td>
<td></td>
<td>inchnstr()</td>
</tr>
<tr>
<td>[mv]win[n]str()</td>
<td>Input a string</td>
<td></td>
<td></td>
<td></td>
<td>innstr()</td>
</tr>
</tbody>
</table>
### The Set of Curses Functions

#### Interface Overview

<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]in[n]ustr()</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>ins_nwstr()</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>
<tr>
<td>[mv][w]printw()</td>
<td>Print formatted output</td>
<td></td>
<td></td>
<td></td>
<td>mvprintw()</td>
</tr>
<tr>
<td>[mv][w]scanw()</td>
<td>Convert formatted output</td>
<td></td>
<td></td>
<td></td>
<td>mvscanw()</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 20). 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 shall be implemented as macros:

<table>
<thead>
<tr>
<th>Macros</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EC</strong> COLOR_PAIR, PAIR_NUMBER()</td>
<td>Refer to can_change_color().</td>
</tr>
<tr>
<td><strong>EC</strong> getbegyx(), getmaxyx(), getparyx(), getyx()</td>
<td>Refer to getbegyx().</td>
</tr>
<tr>
<td><strong>EC</strong> The int arguments passed to getbegyx(), getmaxyx(), getparyx(), and getyx() shall be modifiable lvalues.</td>
<td></td>
</tr>
</tbody>
</table>

---

Technical Standard 2009
3.8 **Initialized 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 initialization 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 7).

To initialize 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:

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

Most programs would additionally use the sequence:

```
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 22).

**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 applications portability at the C-language source level.

The display model defined in Section 3.4 (on page 18) 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 pages must be read in conjunction with this overview information.
NAME
COLOR_PAIRS, COLORS — external variables for color support

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

DESCRIPTION
Refer to `can_change_color()`. 
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.

RETURN VALUE
None.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
initscr(), <curses.h>

CHANGE HISTORY
First released in Issue 4.
NAME
LINES — number of lines on terminal screen

SYNOPSIS
EC
#include <curses.h>
extern int LINES;

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

RETURN VALUE
None.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
initscr(), <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

```c
#include <curses.h>

int add_wch(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);
int wadd_wch(WINDOW *win, 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 special character processing. These functions perform wrapping.

- If `wch` refers to a spacing character, then any previous character at that location is removed, a new character specified by `wch` is placed at that location with rendition specified by `wch`; then the cursor advances to the next spacing character on the screen.

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

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

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Section 3.4.4 (on page 21), `addch()`, `<curses.h>`

CHANGE HISTORY
First released in Issue 4.

Issue 4, Version 2
Corrections made to the SYNOPSIS.
add_wchnstr() 

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 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);
int wadd_wchnstr(WINDOW *win, const cchar_t *wchstr, int n);
int wadd_wchstr(WINDOW *win, 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.

These 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_ts (or the entire array of cchar_ts, if n is −1).

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

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
<curses.h>

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

Corrections made to the SYNOPSIS.
addch( )

CURSES

Curses Interfaces

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 special character processing. These functions perform wrapping.

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

ERRORS
No errors are defined.

EXAMPLES
None.

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.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Section 3.4.4 (on page 21), 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 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(), mvaddchnstr(), mvwaddchnstr(), and waddchnstr() 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.

EXAMPLES

None.

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.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

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

CHANGE HISTORY

First released in Issue 4.

Issue 4, Version 2

Corrections made to the SYNOPSIS.
addnstr()  
NAME
addnstr, addstr, mvaddnstr, mvaddstr, mvwaddnstr, mvwaddstr, waddnstr, waddstr — add a string of multi-byte characters without rendition to a window and advance cursor

SYNOPSIS
```c
#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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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 was changed to indicate that the functions will handle multi-byte sequences correctly.
Issue 4, Version 2
Corrections made to the SYNOPSIS.
NAME
addnwstr, addwstr, mvaddnwstr, mvaddwstr, mvwaddnwstr, mvwaddwstr, waddnwstr, waddwstr — add a wide-character string to a window and advance the cursor

SYNOPSIS
#include <curses.h>

int addnwstr(const wchar_t *wstr, int n);
int addwstr(const wchar_t *wstr);
int mvaddnwstr(int y, int x, const wchar_t *wstr, int n);
int mvaddwstr(int y, int x, const wchar_t *wstr);
int mvwaddnwstr(WINDOW *win, int y, int x, const wchar_t *wstr, int n);
int mvwaddwstr(WINDOW *win, int y, int x, const wchar_t *wstr);
int waddnwstr(WINDOW *win, const wchar_t *wstr, int n);
int waddwstr(WINDOW *win, const wchar_t *wstr);

DESCRIPTION
These functions write the characters of the wide character string wstr on the current or specified window at that window’s current or specified cursor position. These functions advance the cursor position. These functions perform special character processing. These functions perform wrapping. The effect is similar to building a cchar_t from the wchar_t and the background rendition and calling wadd_wch(), once for each wchar_t character in the string. The cursor movement specified by the mv functions occurs only once at the start of the operation.

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

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

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
add_wch(), <curses.h>

CHANGE HISTORY
First released in Issue 4.

Issue 4, Version 2
Corrections made to the SYNOPSIS.
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 color of the window rendition of the current or specified window.

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

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

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

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

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

RETURN VALUE

These functions always return OK.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.
SEE ALSO
   attroff(), <curses.h>

CHANGE HISTORY
   First released in Issue 4.

Issue 4, Version 2
   This entry is rewritten to include the color handling functions wcolor_set() and color_set().
NAME
attroff, attron, attrset, wattroff, wattron, wattrset — restricted window attribute control functions

SYNOPSIS
#include <curses.h>

int attroff(int attrs);
int attron(int attrs);
int attrset(int attrs);
int wattroff(WINDOW *win, int attrs);
int wattron(WINDOW *win, int attrs);
int wattrset(WINDOW *win, int attrs);

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

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

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

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

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

RETURN VALUE
These functions always return either OK or 1.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
Historical implementations returned either OK or 1. This revision allows either behavior.

FUTURE DIRECTIONS
None.

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
  baudrate — get terminal baud rate

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

DESCRIPTION
  The \texttt{baudrate()} function extracts the output speed of the terminal in bits per second.

RETURN VALUE
  The \texttt{baudrate()} function returns the output speed of the terminal.

ERRORS
  No errors are defined.

EXAMPLES
  None.

APPLICATION USAGE
  None.

RATIONALE
  None.

FUTURE DIRECTIONS
  None.

SEE ALSO
  \texttt{tcgetattr()} (in the \texttt{XSH} specification), \texttt{<curses.h>}

CHANGE HISTORY
  First released in Issue 2.

Issue 4
  The argument list is explicitly declared as \texttt{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.

EXAMPLES
   None.

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

RATIONALE
   None.

FUTURE DIRECTIONS
   None.

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
```
#include <curses.h>

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

DESCRIPTION
The `bkgdset()` and `wbkgdset()` functions turn off the previous background attributes, 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, the `bkgd()` and `wbkgd()` functions return OK. Otherwise, they return ERR.

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

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

ERRORS
No errors are defined.

EXAMPLES
None.

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.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Section 3.3.4 (on page 16), `<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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Section 3.3.4 (on page 16), `<curses.h>`
CHANGE HISTORY

First released in Issue 4.

Issue 4, Version 2
Corrections applied.
NAME
border, wborder — draw borders from single-byte characters and renditions

SYNOPSIS
EC
#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</th>
<th>Usage</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ls</td>
<td>Starting-column side</td>
<td>ACS_VLINE</td>
</tr>
<tr>
<td>rs</td>
<td>Ending-column side</td>
<td>ACS_VLINE</td>
</tr>
<tr>
<td>ts</td>
<td>First-line side</td>
<td>ACS_HLINE</td>
</tr>
<tr>
<td>bs</td>
<td>Last-line side</td>
<td>ACS_HLINE</td>
</tr>
<tr>
<td>tl</td>
<td>Corner of the first line and the starting column</td>
<td>ACS_ULCORNER</td>
</tr>
<tr>
<td>tr</td>
<td>Corner of the first line and the ending column</td>
<td>ACS_URCORNER</td>
</tr>
<tr>
<td>bl</td>
<td>Corner of the last line and the starting column</td>
<td>ACS_LLCORNER</td>
</tr>
<tr>
<td>br</td>
<td>Corner of the last line and the ending column</td>
<td>ACS_LRCORNER</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.

EXAMPLES
None.

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.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
border_set(), box(), hline(), <curses.h>
CHANGE HISTORY

First released in Issue 4.

Issue 7

Corrigendum U022/2 is applied, changing the ACS_BLCORNER and ACS_BRCORNER macros to ACS_LLCORNER and ACS_LRCORNER, respectively.
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><code>ls</code></td>
<td>Starting-column side</td>
<td>WACS_VLINE</td>
</tr>
<tr>
<td><code>rs</code></td>
<td>Ending-column side</td>
<td>WACS_VLINE</td>
</tr>
<tr>
<td><code>ts</code></td>
<td>First-line side</td>
<td>WACS_HLINE</td>
</tr>
<tr>
<td><code>bs</code></td>
<td>Last-line side</td>
<td>WACS_HLINE</td>
</tr>
<tr>
<td><code>tl</code></td>
<td>Corner of the first line and the starting column</td>
<td>WACS_ULCORNER</td>
</tr>
<tr>
<td><code>tr</code></td>
<td>Corner of the first line and the ending column</td>
<td>WACS_URCORNER</td>
</tr>
<tr>
<td><code>bl</code></td>
<td>Corner of the last line and the starting column</td>
<td>WACS_LLCORNER</td>
</tr>
<tr>
<td><code>br</code></td>
<td>Corner of the last line and the ending column</td>
<td>WACS_LRCORNER</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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.
SEE ALSO

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

CHANGE HISTORY

First released in Issue 4.

**Issue 4, Version 2**

Corrections made to the SYNOPSIS.

**Issue 7**

Corrigendum U022/2 is applied, changing the WACS_BLCORNER and WACS_BRCORNER macros to WACS_LLCORNER and WACS_LRCORNER, respectively.
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, the box() function returns OK. Otherwise, it returns ERR.

ERRORS
   No errors are defined.

EXAMPLES
   None.

APPLICATION USAGE
   This 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.

RATIONALE
   None.

FUTURE DIRECTIONS
   None.

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, the `box_set()` function returns `OK`. Otherwise, it returns `ERR`.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
`border_set()`, `hline_set()`, `<curses.h>`

CHANGE HISTORY
First released in Issue 4.

Issue 4, Version
 Corrections made to the SYNOPSIS.
NAME
can_change_color, color_content, has_colors, init_color, init_pair, pair_content, start_color —
color manipulation functions

SYNOPSIS

```c
#include <curses.h>

bool can_change_color(void);
int color_content(short color, short *red, short *green, short *blue);
int COLOR_PAIR(int n);
bool has_colors(void);
int init_color(short color, short red, short green, short blue);
int init_pair(short pair, short f, short b);
int pair_content(short pair, short *f, short *b);
int PAIR_NUMBER(int value);
int start_color(void);
extern int COLOR_PAIRS;
extern int COLORS;
```

DESCRIPTION

These functions manipulate color on terminals that support color.

Querying Capabilities

The has_colors() function indicates whether the terminal is a color terminal. The

```
can_change_color() function indicates whether the terminal is a color terminal on which colors
can be redefined.
```

Initialization

The start_color() function must be called in order to enable use of colors and before any color

```
manipulation function is called. The function initializes eight basic colors (black, blue, green,
cyan, red, magenta, yellow, and white) that can be specified by the color macros (such as
COLOR_BLACK) defined in <curses.h> (see Color-Related Macros, on page 309). The initial
appearance of these eight colors is not specified.
```

The function also initializes two global external variables:

- COLORS defines the number of colors that the terminal supports (see Color Identification).
  If COLORS is 0, the terminal does not support redefinition of colors (and
  can_change_color() will return FALSE).

- COLOR_PAIRS defines the maximum number of color-pairs that the terminal supports (see
  User-Defined Color Pairs, on page 57).

The start_color() function also restores the colors on the terminal to terminal-specific initial

```
values. The initial background color is assumed to be black for all terminals.
```

Color Identification

The init_color() function redefines color number color, on terminals that support the redefinition

```
of colors, to have the red, green, and blue intensity components specified by red, green, and blue,
respectively. Calling init_color() also changes all occurrences of the specified color on the screen
to the new definition.
```

The color_content() function identifies the intensity components of color number color. It stores

```
the red, green, and blue intensity components of this color in the addresses pointed to by red, green,
and blue, respectively.
```
For both functions, the color argument must be in the range from 0 to and including COLORS–1. Valid intensity values range from 0 (no intensity component) up to and including 1000 (maximum intensity in that component).

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

After defining the color pair, the macro COLOR_PAIR(n) returns the value of color pair n. This value is the color attribute as it would be extracted from a ctype. Conversely, the macro PAIR_NUMBER(value) returns the color pair number associated with the color attribute value.

The pair_content() function retrieves the component colors of a color-pair number pair. It stores the foreground and background color numbers in the variables pointed to by f and b, respectively.

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

RETURN VALUE
The has_colors() function returns TRUE if the terminal can manipulate colors. Otherwise, it returns FALSE.

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

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

ERRORS
No errors are defined.

EXAMPLES
None.

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

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

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

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
attroff(), delscreen(), <curses.h>
can_change_color()  ENHANCED CURSES  Curses Interfaces

CHANGE HISTORY
First released in Issue 4.

Issue 4, Version
Corrections made in the NAME and APPLICATION USAGE sections.
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.

EXAMPLES

None.

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.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 3.5.2 (on page 23), `<curses.h>`, XBD specification, Chapter 11, General Terminal Interface

CHANGE HISTORY

First released in Issue 2.

Issue 4

The `raw()` and `noraw()` functions are merged with this entry.

The entry is rewritten for clarity.

The argument list for all these functions is explicitly declared as `void`. 
NAME
chgat, mvchgt, 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 mvchgt(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 \( attr \) and \( color \) as for `setcchar()`.

These functions do not update the cursor, except for the initial movement to the specified position by the functions prefixed with \( mv \). These functions do not perform wrapping.

A value of \( n \) that is greater than the remaining characters on a line is not an error.

The \( opts \) argument is reserved for definition in a future version. Currently, the application must provide a null pointer as \( opts \).

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

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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

CHANGE HISTORY
First released in Issue 4.

Issue 4, Version 2
Corrections made to the SYNOPSIS.
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
These 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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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.
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, wsetscrreg — terminal output control functions

SYNOPSIS
#include <curses.h>
int clearok(WINDOW *win, bool bf);
int idlok(WINDOW *win, bool bf);
int leaveok(WINDOW *win, bool bf);
int scrollok(WINDOW *win, bool bf);
int setscrreg(int top, int bot);
int wsetscrreg(WINDOW *win, int top, int bot);

DESCRIPTION
These functions set options that deal with output within Curses.

The clearok() function assigns the value of 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 wsetscrreg() 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, the setscrreg() and wsetscrreg() functions return OK. Otherwise,
they return ERR.

The other functions always return OK.

ERRORS
No errors are defined.
clearok()

**EXAMPLES**
None.

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

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

**RATIONALE**
None.

**FUTURE DIRECTIONS**
None.

**SEE ALSO**
`clear()`, `delscreen()`, `doupdate()`, `scrl()`, `<curses.h>`

**CHANGE HISTORY**
First released in Issue 2.

**Issue 4**
The `idlok()`, `leaveok()`, `scrollok()`, `setscrreg()`, and `wsetscrreg()` functions are merged with this entry.

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 `curser` or the specified window.

The RETURN VALUE section 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
These 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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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

These 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.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

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 color

SYNOPSIS

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

DESCRIPTION
Refer to `can_change_color()`. 
NAME
color_set — window attribute control functions

SYNOPSIS
#include <curses.h>
int color_set(short color_pair_number, void *opts);

DESCRIPTION
Refer to attr_get().
NAME

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, this function returns OK. Otherwise, it returns ERR.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

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

CHANGE HISTORY

First released in Issue 4.

Corrections made to the SYNOPSIS.
NAME
cur_term — current terminal information

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

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

RETURN VALUE
None.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
`del_curterm()`, `tigetflag()`, `<term.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 <code>visibility</code></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 the `curs_set()` function returns the previous cursor state. Otherwise, it returns ERR.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

`<curses.h>`

CHANGE HISTORY

First released in Issue 4.
NAME
curscr — current window

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

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

RETURN VALUE
None.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
clearok(), <curses.h>

CHANGE HISTORY
First released in Issue 4.
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.

EXAMPLES

None.

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.

RATIONALE

None.

FUTURE DIRECTIONS

None.

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.

The entry is rewritten for clarity.

The argument list for all these functions 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, the setupterm() function must be called first. It is automatically called by initscr() and newterm(). The setupterm() function initializes the other functions to use the terminfo record for a specified terminal (which depends on whether use_env() was called). It sets the cur_term external variable to a TERMINAL structure that contains the record from the terminfo database for the specified terminal.

The terminal type is the character string term; if term is a null pointer, the environment variable TERM is used. If TERM is not set or if its value is an empty string, then unknown is used as the terminal type. The application must set fildes to a file descriptor, open for output, to the terminal device, before calling setupterm(). If errret is not null, the integer it points to is set to one of the following values to report the function outcome:

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

If setupterm() detects an error and errret is a null pointer, the setupterm() function 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, the set_curterm() function 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.

EXAMPLES
None.

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

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Section A.3 (on page 377), `baudrate()`, `erasechar()`, `has_ic()`, `longname()`, `termattrs()`, `termname()`, `tigetflag()`, `use_env()`, `<term.h>`

CHANGE HISTORY
First released in Issue 4.
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, the delay_output() function returns OK. Otherwise, it returns ERR.

ERRORS
No errors are defined.

EXAMPLES
None.

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

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Section 7.1.3 (on page 340), 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

```c
#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. These functions do not change the cursor position.

RETURN VALUE

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

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

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
deleteln, wdeleteln — delete lines in a window

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

DESCRIPTION
These 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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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 deleteln() function is explicitly declared as void.
NAME
delscreen — free storage associated with a screen

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

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

RETURN VALUE
This function does not return a value.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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
This function deletes win, freeing all memory associated with it. The application must delete
subwindows before deleting the main window.

RETURN VALUE
Upon successful completion, the delwin() function returns OK. Otherwise, it returns ERR.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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 size of a window cannot be greater than the physical size of the screen, or that defined using the environment variables `LINES` and `COLUMNS`. The behavior of a window which extends outside the terminal screen is undefined.

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.

EXAMPLES

None.

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:
```c
newwin(0, 0, 0, 0);
```

Pads should be used whenever a window larger than the terminal screen is required.

RATIONALE

None.

FUTURE DIRECTIONS

None.
SEE ALSO

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

CHANGE HISTORY

First released in Issue 4.

Issue 7

Corrigendum U018/4 is applied, adding window size to the description of the newwin() function, and adding use of pads to the APPLICATION USAGE section.
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.

If the win parameter to wrefresh() is equal to the value of curscr, the screen is immediately
cleared and repainted.

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.

EXAMPLES
None.

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.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
clearok(), curscr, redrawwin(), <curses.h>

CHANGE HISTORY
First released in Issue 4.

This entry is a merge 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
This function creates a duplicate of the window win.

RETURN VALUE
Upon successful completion, the dupwin() function returns a pointer to the new window. Otherwise, it returns a null pointer.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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 behavior is undefined.

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

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Section 3.5 (on page 22), getch(), <curses.h>, XBD specification, Section 11.2, 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
The state of the echo modes is further clarified.
NAME

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

SYNOPSIS

```c
#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.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

`addch()`, `add_wch()`, `doupdate()`, `<curses.h>`

CHANGE HISTORY

First released in Issue 4.

Issue 4, Version 2

Corrections made to the SYNOPSIS.
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.

EXAMPLES
None.

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.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
addch(), douupdate(), echo wchar(), <curses.h>

CHANGE HISTORY
First released in Issue 4.
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, the endwin() function returns OK. Otherwise, it returns ERR.

ERRORS
No errors are defined.

EXAMPLES
None.

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().

RATIONALE
None.

FUTURE DIRECTIONS
None.

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.
erased()  CURSES  Curses Interfaces

NAME
erase, werase — clear a window

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

DESCRIPTION
Refer to clear().
NAME
erasechar, erasewchar, killchar, killwchar — terminal environment query functions

SYNOPSIS

```c
#include <curses.h>

char erasechar(void);

int erasewchar(wchar_t *ch);

char killchar(void);

int killwchar(wchar_t *ch);
```

DESCRIPTION

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

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

RETURN VALUE

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

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

ERRORS

No errors are defined.

EXAMPLES

None.

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.

RATIONALE

None.

FUTURE DIRECTIONS

None.

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 entry is rewritten for clarity.

The argument list for the `erasechar()` and `killchar()` functions is explicitly declared as `void`.
The `erasechar()` and `killwchar()` functions 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 initializing terminal capabilities that assume that
the terminal has more than one line. A subsequent call to initscr() or newterm() performs the
following additional actions:

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

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

RETURN VALUE
The filter() function does not return a value.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
  Section 7.1.3 (on page 340), 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.

EXAMPLES
    None.

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

RATIONALE
    None.

FUTURE DIRECTIONS
    None.

SEE ALSO
    beep(), <curses.h>

CHANGE HISTORY
    First released in Issue 4.

    In previous versions, 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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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

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

SYNOPSIS

```
#include <curses.h>

int get_wch(wint_t *ch);
int mvget_wch(int y, int x, wint_t *ch);
int mvwget_wch(WINDOW *win, int y, int x, wint_t *ch);
int wget_wch(WINDOW *win, wint_t *ch);
```

DESCRIPTION

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

Processing of terminal input is subject to the general rules described in Section 3.5 (on page 22).
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 20) 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 (`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
application may produce undesirable results.

RATIONALE

None.
FUTURE DIRECTIONS
None.

SEE ALSO
Section 3.5 (on page 22), beep(), cbreak(), ins_wch(), Section A.1.8, move(), <curses.h>, <wchar.h>
(in the XBD specification)

CHANGE HISTORY
First released in Issue 4.
**get_wstr()**

**NAME**
get_wstr — get an array of wide characters and function key codes from a terminal

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

int get_wstr(wint_t *wstr);
```

**DESCRIPTION**
Refer to `getn_wstr()`.
NAME
getbegyx, getmaxyx, getparyx, getyx — get cursor and window coordinates

SYNOPSIS

```c
#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`.

The application shall ensure that the `y` and `x` arguments are modifiable lvalues.

RETURN VALUE
No return values are defined.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
Historical implementations often defined the following functions (which may have been implemented as macros):

```c
int getbegx(WINDOW *win);
int getbegy(WINDOW *win);
int getcurx(WINDOW *win);
int getcury(WINDOW *win);
int getmaxx(WINDOW *win);
int getmaxy(WINDOW *win);
int getparx(WINDOW *win);
int getpary(WINDOW *win);
```

Although `getbegyx()`, `getyx()`, `getmaxyx()`, and `getparyx()` provide the required functionality, this does not preclude applications from defining these functions for their own use. For example, to implement:

```c
int getbegx(WINDOW *win);
```

a suitable function would be:

```c
int getbegx(WINDOW *win)
{
    int x, y;
    getbegyx(win, y, x);
    return x;
}
```
RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
<curses.h>

CHANGE HISTORY
First released in Issue 4.

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

SYNOPSIS

```
#include <curses.h>

cttype getbkgd(WINDOW *win);
```

DESCRIPTION
Refer to `bkgd()`.
NAME
getbkgrnd — get background character and rendition

SYNOPSIS
```
#include <curses.h>

int getbkgrnd(cchar_t *ch);
```

DESCRIPTION
Refer to \textit{bkgrnd(}).
NAME
getcchar — get a wide-character string and rendition from a cchar_t

SYNOPSIS

#include <curses.h>

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

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

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

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

RETURN VALUE
When wch is a null pointer, the getcchar() function returns the number of wide characters referenced by wcval, including the null terminator.

When wch is not a null pointer, the getcchar() function returns OK upon successful completion. Otherwise, it returns ERR.

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.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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

CHANGE HISTORY
First released in Issue 4.

Issue 4, Version 2
Corrections made to the SYNOPSIS.
getch()eties to 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 22).
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 20) 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, these functions return the single-byte character, KEY_ value, or
ERR. When in the nodelay mode (nodelay()) and no data is available, ERR is returned.

ERRORS
No errors are defined.

EXAMPLES
None.

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.

RATIONALE
None.

FUTURE DIRECTIONS
None.
SEE ALSO

Section 3.5 (on page 22), cbreak(), doupdate(), 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 as `void`.

Issue 4, Version 2
The RETURN VALUE section is expanded.
getmaxyx() (ENHANCED CURSES)

NAME
getmaxyx — get size of a window

SYNOPSIS

```c
#include <curses.h>

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

DESCRIPTION
Refer to `getbegyx()`.
NAME

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

SYNOPSIS

```c
#include <curses.h>

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

DESCRIPTION

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

The user’s erase and kill characters are interpreted and affect the sequence of characters returned.

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

The effect of `mvget_wstr()` is as though a call to `move()` followed by a series of calls to `get_wch()` were made. The effect of `mvwget_wstr()` is as though a call to `wmove()` followed by a series of calls to `wget_wch()` were made. The effect of `mvwgetn_wstr()` is as though a call to `wmove()` followed by 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.

EXAMPLES

None.

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.

RATIONALE

None.
**getn_wstr()**

**ENHANCED CURSES**

**Curses Interfaces**

---

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

`get_wch()`, `getnstr()`, `<curses.h>`, `<wchar.h>` (in the XBD specification), XBD specification, Chapter 11, General Terminal Interface

**CHANGE HISTORY**

First released in Issue 4.

**Issue 7**

Corrigendum U018/1 is applied, correcting the `getn_wstr()` and `get_wstr()` function prototypes.
NAME
getnstr, getstr, mvgetnstr, mvgetstr, mvwgetnstr, mvwgetstr, wgetnstr, wgetstr — get a multibyte 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 are equivalent to the getstr(), mvgetstr(), mvwgetstr(), and wgetstr() functions respectively, except that they read at most n-1 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() followed by a series of calls to getch(). The mvwgetstr() function is identical to getstr() except it is as though a call to wmove() is made followed by a series of calls to wgetch(). The move() function is identical to getnstr() except that it is as though it is a call to move() followed by a series of calls to getch(). The mvwgetnstr() function is identical to getnstr() except it is as though a call to wmove() is made followed by 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.

EXAMPLES
None.

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.

RATIONALE
None.

FUTURE DIRECTIONS
None.
SEE ALSO

Section 3.5 (on page 22), 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.

Issue 4, Version 2

Corrections made to first sentence of the DESCRIPTION.
NAME

getparyx — get subwindow origin coordinates

SYNOPSIS

```c
#include <curses.h>

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

DESCRIPTION

Refer to `getbegyx()`.
NAME
    getstr — get a multi-byte character string from the terminal

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

DESCRIPTION
    Refer to getnstr().
NAME
getwin, putwin — dump window to, and reload window from, a file

SYNOPSIS

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

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

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

RETURN VALUE
Upon successful completion, the `getwin()` function returns a pointer to the window it created. Otherwise, it returns a null pointer.

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

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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

CHANGE HISTORY
First released in Issue 4.
getyx() CURSES

NAME
getyx — get cursor coordinates

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

DESCRIPTION
Refer to 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, the halfdelay() function returns OK. Otherwise, it returns ERR.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
The application can call nocbreak() to leave Half-Delay mode.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Section 3.5.2 (on page 23), cbreak(), <curses.h>, XBD specification, Chapter 11, General Terminal Interface

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

SYNOPSIS
EC
#include <curses.h>

bool has_colors(void);

DESCRIPTION
Refer to can_change_color().
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-character and delete-character
   capabilities.

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

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

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

ERRORS
   No errors are defined.

EXAMPLES
   None.

APPLICATION USAGE
   The has_il() function may be used to determine whether it would be appropriate to turn on
   physical scrolling using scrollok().

RATIONALE
   None.

FUTURE DIRECTIONS
   None.

SEE ALSO
   <curses.h>

CHANGE HISTORY
   First released in Issue 2.

   Issue 4
   The has_il() function is merged with this entry.
   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
#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.

EXAMPLES
None.

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.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
border_set(), <curses.h>

CHANGE HISTORY
First released in Issue 4.

Issue 4, Version 2
Corrections made to the SYNOPSIS.
NAME
idcok — enable or disable use of hardware insert-character and delete-character features

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

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

DESCRIPTION
The idcok() function specifies whether the implementation may use hardware insert-character 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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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

CHANGE HISTORY
First released in Issue 4.
NAME
idlok — enable or disable use of terminal insert-character and delete-line features

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

DESCRIPTION
Refer to 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.

EXAMPLES
None.

APPLICATION USAGE
The `immedok()` function is useful for windows that are used as terminal emulators.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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

CHANGE HISTORY
First released in Issue 4.
NAME
in_wch, mvin_wch, mvwin_wch, win_wch — extract a complex character and rendition from a window

SYNOPSIS
#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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
<curses.h>

CHANGE HISTORY
First released in Issue 4.
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

#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() functions 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.

EXAMPLES

None.

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.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

in_wch(), <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>

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

DESCRIPTION

    These functions return the character and rendition, of type ctype, 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 (ctype)ERR.

ERRORS

    No errors are defined.

EXAMPLES

    None.

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.

RATIONALE

    None.

FUTURE DIRECTIONS

    None.

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, mvinchnstr, mvinchstr, 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 mvinchnstr(int y, int x, chtype *chstr, int n);
int mvinchstr(int y, int x, chtype *chstr);
int mvwinchnstr(WINDOW *win, int y, int x, chtype *chstr, int n);
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(), mvinchnstr(), mvwinchnstr(), and winchnstr() 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.

EXAMPLES

None.

APPLICATION USAGE

Reading a line that overflows the array pointed to by chstr with inchstr(), mvinchstr(), mvwinchstr(), or winchstr() causes undefined results. The use of inchnstr(), mvinchnstr(), mvwinchnstr(), or winchnstr(), respectively, is recommended.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

inch(), <curses.h>

CHANGE HISTORY

First released in Issue 4.
NAME
init_color, init_pair — redefine specified color or color pair

SYNOPSIS
```
#include <curses.h>

int init_color(short color, short red, short green, short blue);
int init_pair(short pair, short f, short b);
```

DESCRIPTION
Refer to `can_change_color()`.
NAME
initscr, newterm — screen initialization functions

SYNOPSIS
#include <curses.h>
WINDOW *initscr(void);
SCREEN *newterm(const char *type, FILE *outfile, FILE *infile);

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

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

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

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

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

RETURN VALUE
Upon successful completion, the initscr() function returns a pointer to stdscr. Otherwise, it does
not return.

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

ERRORS
No errors are defined.

EXAMPLES
None.

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
signa l s before calling initscr().
RATIONAL
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Section A.3 (on page 377), delscreen(), douupdate(), del_curterm(), filter(), slk_attroff(), use_env(), <curses.h>

CHANGE HISTORY
First released in Issue 2.

Issue 4
The newterm() function is merged with this entry.
The entry is rewritten for clarity.
The argument list for the initscr() function is explicitly declared as void.

Issue 7
The prototype for the newterm() function is updated.
innstr() ENHANCED CURSES

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

SYNOPSIS

```c
#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 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, the `instr()`, `mvinstr()`, `mvwinstr()`, and `winstr()` functions return OK.

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

Otherwise, all these functions return ERR.

ERRORS

No errors are defined.

EXAMPLES

None.

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()`, `mvwinstr()`, or `winstr()` causes undefined results. The use of `innstr()`, `mvinnstr()`, `mvwinnstr()`, or `winnstr()`, respectively, is recommended.

RATIONALE

None.
FUTURE DIRECTIONS
None.

SEE ALSO
<curses.h>

CHANGE HISTORY
First released in Issue 4.
**NAME**
innwstr, inwstr, mvinnwstr, mvinwstr, mvwinnwstr, mvwinwstr, winnwstr, winwstr — 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, the inwstr(), mvinwstr(), mvwinwstr(), and winwstr() functions return OK.

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

Otherwise, all these functions return ERR.

**ERRORS**
No errors are defined.

**EXAMPLES**
None.

**APPLICATION USAGE**
Reading a line that overflows the array pointed to by wstr with inwstr(), mvinwstr(), mvwinnwstr(), or winwstr() causes undefined results. The use of innwstr(), mvinnwstr(), mvwinnwstr(), or winwstr(), respectively, is recommended.

These functions do not return rendition information.

**RATIONALE**
None.

**FUTURE DIRECTIONS**
None.
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
C
#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 advance the cursor position. These functions perform special character
processing. These functions do not perform wrapping.

The ins_nwstr(), mvins_nwstr(), mvwins_nwstr(), and wins_nwstr() functions insert at most n
wchar_t characters. If n is less than 0, then the entire string is inserted.

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

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
<curses.h>

CHANGE HISTORY
First released in Issue 4.

Issue 4, Version 2
Corrections made to the SYNOPSIS.
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 mvins_wch(int y, int x, const cchar_t *wch);
int mvwins_wch(WINDOW *win, int y, int x, const cchar_t *wch);
int wins_wch(WINDOW *win, 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 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. These functions do not perform wrapping.

RETURN VALUE

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

ERRORS

No errors are defined.

EXAMPLES

None.

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.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

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

CHANGE HISTORY

First released in Issue 4.

Issue 4, Version 2

Corrections made to the SYNOPSIS.
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 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. These functions do not perform wrapping.

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

ERRORS
No errors are defined.

EXAMPLES
None.

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.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
ins_wch(), <curses.h>

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity.

Issue 4, Version 2
The DESCRIPTION is further clarified.
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
These 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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
deletein(), insertln(), <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
These 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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
insdeIln(), <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. These 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.

EXAMPLES
None.

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.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
`<curses.h>`

CHANGE HISTORY
First released in Issue 4.
Issue 4, Version 2
Corrections made to the SYNOPSIS.
instr()  ENHANCED CURSES  Curses Interfaces

NAME
instr — input a multi-byte character string from the current window

SYNOPSIS
EC
#include <curses.h>
int instr(char *str);

DESCRIPTION
Refer to instr().
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, the intrflush() function returns OK. Otherwise, it returns ERR.

ERRORS
No errors are defined.

EXAMPLES
None.

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

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Section 3.5 (on page 22), <curses.h>, XBD specification, Section 11.2, 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
inwstr — input a string of wide characters from the current window

SYNOPSIS
EC
#include <curses.h>
int inwstr(wchar_t *wstr);

DESCRIPTION
Refer to inwstr().
NAME

is_linetouched, is_wintouched, touchline, touchwin, untouchwin, wtouchn — 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);
EC int untouchwin(WINDOW *win);
int wtouchn(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 `wtouchn()`, if `changed` is 1, touches `n` lines in the specified window, starting at line `y`. If `changed` is 0, `wtouchn()` 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.

EXAMPLES

None.

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.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

`Section 3.2 (on page 14), douptate(), <curses.h>`
is_linetouched()

CHANGE HISTORY
First released in Issue 4.
NAME
isendwin — determine whether a screen has been refreshed

SYNOPSIS

```c
#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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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

CHANGE HISTORY
First released in Issue 4.
keyname()  

NAME  
keyname, key_name — get name of key  

SYNOPSIS  
```  
#include <curses.h>  
char *keyname(int c);  
char *key_name(wchar_t c);  
```

DESCRIPTION  
These 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>ˆX</td>
</tr>
<tr>
<td>Meta-character (keyname() only)</td>
<td>−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, these functions return a pointer to a string as described above. Otherwise, they return a null pointer.

ERRORS  
No errors are defined.

EXAMPLES  
None.

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.

RATIONALE  
None.

FUTURE DIRECTIONS  
None.

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 behavior of any function that provides keyboard input.

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

RETURN VALUE
Upon successful completion, the keypad() function returns OK. Otherwise, it returns ERR.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Section 3.5.1 (on page 22), <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().
NAME
leaveok — control cursor position resulting from refresh operations

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

DESCRIPTION
Refer to clearok().
leavok() CURSES Curses Interfaces

NAME
  leavok — terminal output control functions

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

DESCRIPTION
  Refer to clearok().
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, the longname() function returns a pointer to the description
   specified above. Otherwise, it returns a null pointer on error.

ERRORS
   No errors are defined.

EXAMPLES
   None.

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

RATIONALE
   None.

FUTURE DIRECTIONS
   None.

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 seven or eight significant bits on input depends on the control mode of the display driver (see the XBD specification, General Terminal Interface). To force eight bits to be returned, invoke meta(win, TRUE). To force seven 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, the meta() function returns OK. Otherwise, it returns ERR.

ERRORS
No errors are defined.

EXAMPLES
None.

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.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Section 3.5 (on page 22), getch(), <curses.h>, XBD specification, Section 11.2, 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
These 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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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.)

Note: 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 meprintw() 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.

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(), mvwaddch()</td>
<td>addch()</td>
</tr>
<tr>
<td>mvaddchnstr(), mvwaddchnstr()</td>
<td>addchstr()</td>
</tr>
<tr>
<td>mvaddchstr(), mvwaddchstr()</td>
<td>addchstr()</td>
</tr>
<tr>
<td>mvaddnstr(), mvwaddnstr()</td>
<td>addnstr()</td>
</tr>
<tr>
<td>mvaddstr(), mvwaddstr()</td>
<td>addstr()</td>
</tr>
<tr>
<td>mvaddnstr(), mvwaddnstr()</td>
<td>addnstr()</td>
</tr>
<tr>
<td>mvadd_wch(), mvwadd_wch()</td>
<td>add_wch()</td>
</tr>
<tr>
<td>mvadd_wchnstr(), mvwadd_wchnstr()</td>
<td>add_wchnstr()</td>
</tr>
<tr>
<td>mvadd_wchstr(), mvwadd_wchstr()</td>
<td>add_wchstr()</td>
</tr>
<tr>
<td>mvchgt(), mvwchgt()</td>
<td>chgt()</td>
</tr>
<tr>
<td>mvdelch(), mvwdelch()</td>
<td>delch()</td>
</tr>
<tr>
<td>mvgetch(), mvwgetch()</td>
<td>getch()</td>
</tr>
<tr>
<td>mvgetnstr(), mvwgetnstr()</td>
<td>getnstr()</td>
</tr>
<tr>
<td>mvgetstr(), mvwgetstr()</td>
<td>getnstr()</td>
</tr>
<tr>
<td>mvget_ustr(), mvwget_ustr()</td>
<td>get_ustr()</td>
</tr>
<tr>
<td>mvget_wch(), mvwget_wch()</td>
<td>get_wch()</td>
</tr>
<tr>
<td>mvget_ustr(), mvwget_ustr()</td>
<td>get_ustr()</td>
</tr>
<tr>
<td>mvhline(), mvwhline()</td>
<td>hline()</td>
</tr>
<tr>
<td>mvhline_set(), mvwhline_set()</td>
<td>hline_set()</td>
</tr>
<tr>
<td>mvinch(), mvwinch()</td>
<td>inch()</td>
</tr>
<tr>
<td>mvinchnstr(), mvwinchnstr()</td>
<td>inchnstr()</td>
</tr>
<tr>
<td>mvinchstr(), mvwinchstr()</td>
<td>inchstr()</td>
</tr>
<tr>
<td>mvinnstr(), mvwinnstr()</td>
<td>innstr()</td>
</tr>
<tr>
<td>mvinnwstr(), mvwinnwstr()</td>
<td>innwstr()</td>
</tr>
<tr>
<td>mvinsch(), mvwinsch()</td>
<td>insch()</td>
</tr>
<tr>
<td>mvinsnstr(), mvwinsnstr()</td>
<td>insnstr()</td>
</tr>
<tr>
<td>mvinsstr(), mvwinsstr()</td>
<td>insnstr()</td>
</tr>
<tr>
<td>mvinsnstr(), mvwinsnstr()</td>
<td>insnstr()</td>
</tr>
<tr>
<td>mvins_ustr(), mvwins_ustr()</td>
<td>ins_ustr()</td>
</tr>
<tr>
<td>mvins_wch(), mvwins_wch()</td>
<td>ins_wch()</td>
</tr>
<tr>
<td>mvins_ustr(), mvwins_ustr()</td>
<td>ins_ustr()</td>
</tr>
<tr>
<td>mvinswstr(), mvwinwstr()</td>
<td>inswstr()</td>
</tr>
<tr>
<td>mvinswstr(), mvwinwstr()</td>
<td>inswstr()</td>
</tr>
<tr>
<td>Function</td>
<td>Refer to</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>mvin_wch()</code>, <code>mvwin_wch()</code></td>
<td><code>in_wch()</code></td>
</tr>
<tr>
<td><code>mvin_wchnstr()</code>, <code>mvwin_wchnstr()</code></td>
<td><code>in_wchnstr()</code></td>
</tr>
<tr>
<td><code>mvin_wchstr()</code>, <code>mvwin_wchstr()</code></td>
<td><code>in_wchstr()</code></td>
</tr>
<tr>
<td><code>mvprintw()</code>, <code>mvwprintw()</code></td>
<td><code>mvprintw()</code></td>
</tr>
<tr>
<td><code>mvs Camb()</code>, <code>mvwscamw()</code></td>
<td><code>mvs Camb()</code></td>
</tr>
<tr>
<td><code>mvchline()</code>, <code>mvchline()</code></td>
<td><code>hline()</code></td>
</tr>
<tr>
<td><code>mvchline_set()</code>, <code>mvwchline_set()</code></td>
<td><code>hline_set()</code></td>
</tr>
</tbody>
</table>
NAME
mvadd_wch, mvwadd_wch — add a complex character and rendition to a window

SYNOPSIS
#include <curses.h>

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
Refer to add_wch().
NAME

mvadd_wchnstr, mvadd_wchstr, mvwadd_wchnstr, mvwadd_wchstr — add an array of complex characters and renditions to a window

SYNOPSIS

```c
#include <curses.h>

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

Refer to `add_wchnstr()`. 
NAME
mvaddch, mvwaddch — add a single-byte character and rendition to a window and advance the
cursor

SYNOPSIS
#include <curses.h>

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

DESCRIPTION
Refer to addch().
NAME
mvaddchstr, mvaddchnstr, mvwaddchstr, mvwaddchnstr — add string of single-byte characters and renditions to a window

SYNOPSIS
#include <curses.h>

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);

DESCRIPTION
Refer to addchstr().
NAME
mvaddnstr, mvaddstr, mvwaddnstr, mvwaddstr — add a string of multi-byte characters without
rendition to a window and advance cursor

SYNOPSIS
#include <curses.h>

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);

DESCRIPTION
Refer to addnstr().
NAME
mvaddnwstr, mvaddwstr, mvwaddnwstr, mvwaddwstr — add a wide-character string to a window and advance the cursor

SYNOPSIS
#include <curses.h>

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);

DESCRIPTION
Refer to addnwstr().
mvchgt()  ENHANCED CURSES  Curses Interfaces

NAME
mvchgt, mvwchgt — change renditions of characters in a window

SYNOPSIS

```c
#include <curses.h>

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

DESCRIPTION

Refer to chgt().
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 moves 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, the mvcur() function returns OK. Otherwise, it returns ERR.

ERRORS
No errors are defined.

EXAMPLES
None.

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.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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

CHANGE HISTORY
First released in Issue 4.
**mvdelch()**  

**NAME**

mvdelch, mvwdelch — delete a character from a window

**SYNOPSIS**

```c
#include <curses.h>

int mvdelch(int y, int x);
int mvwdelch(WINDOW *win, int y, int x);
```

**DESCRIPTION**

Refer to `delch()`.
NAME

mvderwin — define window coordinate transformation

SYNOPSIS

```c
#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 \((\text{par\_y}, \text{par\_x})\) 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, the `mvderwin()` function returns OK. Otherwise, it returns ERR.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

`derwin()`, `doupdate()`, `dupwin()`, `<curses.h>`

CHANGE HISTORY

First released in Issue 4.
mvget_wch() ENHANCED CURSES

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

SYNOPSIS

```c
#include <curses.h>

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

DESCRIPTION
Refer to get_wch().
NAME
mvgetch, mvwgetch — get a single-byte character from the terminal

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

DESCRIPTION
Refer to getch().
NAME
mvgetn_wstr, mvget_wstr, mvwgetn_wstr, mvwget_wstr — get an array of wide characters and
function key codes from a terminal

SYNOPSIS

```c
#include <curses.h>

int mvgetn_wstr(int y, int x, wint_t *wstr, int n);
int mvget_wstr(int y, int x, wint_t *wstr);
int mvwgetn_wstr(WINDOW *win, int y, int x, wint_t *wstr, int n);
int mvwget_wstr(WINDOW *win, int y, int x, wint_t *wstr);
```

DESCRIPTION
Refer to `getn_wstr()`.
NAME
mvgetnstr, mvgetstr, mvwgetnstr, mvwgetstr — get a multi-byte character string from the terminal

SYNOPSIS
#include <curses.h>

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);

DESCRIPTION
Refer to getnstr().
NAME
mvhline, mvvline, mvwhline, mvwvline — draw lines from single-byte characters and renditions

SYNOPSIS

```c
#include <curses.h>

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);
```

DESCRIPTION

Refer to `hline()`. 
**NAME**

mvhline_set, mvvline_set, mvwhline_set, mvwvline_set — draw lines from complex characters and renditions

**SYNOPSIS**

```c
#include <curses.h>

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);
```

**DESCRIPTION**

Refer to `hline_set()`.

X/Open Curses, Issue 7
NAME
mvin_wch, mvwin_wch — extract a complex character and rendition from a window

SYNOPSIS
#include <curses.h>

int mvin_wch(int y, int x, cchar_t *wcval);
int mvwin_wch(WINDOW *win, int y, int x, cchar_t *wcval);

DESCRIPTION
Refer to in_wch().
mvin_wchnstr( ), mvin_wchstr, mvwin_wchnstr, mvwin_wchstr — extract an array of complex characters and renditions from a window

DESCRIPTION
Refer to in_wchnstr().
**NAME**

mvinch, mvwinch — input a single-byte character and rendition from a window

**SYNOPSIS**

```c
#include <curses.h>

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

**DESCRIPTION**

Refer to `inch()`.
NAME
mvinchnstr, mvinchstr, mvwinchnstr, mvwinchstr — input an array of single-byte characters and renditions from a window

SYNOPSIS

```
#include <curses.h>

int mvinchnstr(int y, int x, chtype *chstr, int n);
int mvinchstr(int y, int x, chtype *chstr);
int mvwinchnstr(WINDOW *win, int y, int x, chtype *chstr, int n);
int mvwinchstr(WINDOW *win, int y, int x, chtype *chstr);
```

DESCRIPTION
Refer to `inchnstr()`.
mvinnstr()  ENHANCED CURSES  Curses Interfaces

NAME
mvinnstr, mvinstr, mvwinnstr, mvwinstr — input a multi-byte character string from a window

SYNOPSIS

```
#include <curses.h>

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);
```

DESCRIPTION

Refer to `innstr()`.
NAME

mvinnwstr, mvinwstr, mvwinnwstr, mvwinwstr — input a string of wide characters from a window

SYNOPSIS

```c
#include <curses.h>

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);
```

DESCRIPTION

Refer to `innwstr()`. 
mvins_nwstr()  ENHANCED CURSES  Curses Interfaces

NAME

mvins_nwstr, mvins_wstr, mvwins_nwstr, mvwins_wstr — insert a wide-character string into a window

SYNOPSIS

```c
#include <curses.h>

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);
```

DESCRIPTION

Refer to ins_nwstr().
NAME
mvins_wch, mvwins_wch — insert a complex character and rendition into a window

SYNOPSIS
#include <curses.h>

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
Refer to ins_wch().
mvinsch() CURSES

NAME
mvinsch, mvwinsch — insert a single-byte character and rendition into a window

SYNOPSIS
#include <curses.h>

int mvinsch(int y, int x, chtype ch);

int mvwinsch(WINDOW *win, int y, int x, chtype ch);

DESCRIPTION
Refer to insch().
NAME
mvinsnstr, mvinsstr, mvwinsnstr, mvwinsstr — insert a multi-byte character string into a window

SYNOPSIS
```
#include <curses.h>

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);
```

DESCRIPTION
Refer to insnstr().
NAME
mvprintw, mvwprintw, printw, wprintw — print formatted output in window

SYNOPSIS
#include <curses.h>

int mvprintw(int y, int x, const char *fmt, ...);
int mvwprintw(WINDOW *win, int y, int x, const char *fmt, ...);
int printw(const char *fmt, ...);
int wprintw(WINDOW *win, const char *fmt, ...);

DESCRIPTION
These 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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
addinstr(), 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, const char *fmt, ...);
int mvwscanw(WINDOW *win, int y, int x, const char *fmt, ...);
int scanw(const char *fmt, ...);
int wscanw(WINDOW *win, const 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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
fscanf() (in the XSH specification), getnstr(), mvprintw(), 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().

Issue 7
The prototypes for the mvscanw(), mvwscanw(), scanw(), and wscanw() functions are updated.
mvwin() CURSES Curses Interfaces

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, the mvwin() function returns OK. Otherwise, it returns ERR.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
The application should not move subwindows by calling mvwin(). Moving subwindows may cause processing in other subwindows in the parent window to become confused if the new location of the subwindow overlays or reveals part of another subwindow.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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.

EXAMPLES
   None.

APPLICATION USAGE
   A more reliable method of achieving a timed delay is the nanosleep() function.

RATIONALE
   None.

FUTURE DIRECTIONS
   None.

SEE ALSO
   delay_output(), nanosleep() (in the XSH specification), <curses.h>

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

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

SYNOPSIS
#include <curses.h>
WINDOW *newpad(int nlines, int ncols);
int pnoutrefresh(WINDOW *pad, int pminrow, int pmincol, int sminrow,
    int smincol, int smaxrow, int smaxcol);
int prefresh(WINDOW *pad, int pminrow, int pmincol, int sminrow,
    int smincol, int smaxrow, int smaxcol);
WINOD *subpad(WINDOW *orig, int nlines, int ncols, int begin_y,
    int begin_x);

DESCRIPTION
The newpad() function creates a specialized window called a pad with nlines lines and ncols
columns. A pad is like a window, except that it is not restricted by the screen size and is not
necessarily associated with a particular part of the screen. Automatic refreshes of pads (e.g.,
from scrolling or echoing of input) do not occur.

The subpad() function creates a specialized window within a pad (called the parent pad) called a
subpad with nlines lines and ncols columns. Unlike subwin(), which uses screen coordinates, the
subpad is created at position (begin_y, begin_x) within the parent pad. Changes made to either
the parent or the subpad affect the other. The subpad must fit totally within the parent pad.

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, the pnoutrefresh() and prefresh() functions return OK. Otherwise,
they return ERR.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
To refresh a pad, call prefresh() or pnoutrefresh(), not wrefresh().

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().

Pads should be used whenever a window larger than the terminal screen is required.
RATIONALE
None.

FUTURE DIRECTIONS
None.

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

CHANGE HISTORY
First released in Issue 2.

Issue 4
The pnoutrefresh() and prefresh() functions are merged with this entry.
The subpad() function is new in Issue 4.

Issue 7
Corrigendum U018/4 is applied, updating the DESCRIPTION of the newpad() and subpad() functions and adding use of pads to the APPLICATION USAGE section.
newterm()  CURSES  Curses Interfaces

NAME
newterm — screen initialization function

SYNOPSIS
#include <curses.h>
SCREEN *newterm(const char *type, FILE *outfile, FILE *infile);

DESCRIPTION
Refer to initscr().
newwin — create a new window

SYNOPSIS

```c
#include <curses.h>

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

DESCRIPTION

Refer to `derwin()`. 
NAME
nl, nonl — enable/disable newline translation

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

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

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

ERRORS
No errors are defined.

EXAMPLES
None.

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

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
<curses.h>

CHANGE HISTORY
First released in Issue 2.

Issue 4
The entry is rewritten for clarity.
The argument list for the nl() and nonl() functions is explicitly declared as 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.

Note: 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.

They are found in 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>
nocbreak()  

NAME
nocbreak, noraw — input mode control functions

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

DESCRIPTION
Refer to cbreak().
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, the nodelay() function returns OK. Otherwise, it returns ERR.

ERRORS
  No errors are defined.

EXAMPLES
  None.

APPLICATION USAGE
  None.

RATIONALE
  None.

FUTURE DIRECTIONS
  None.

SEE ALSO
  Section 3.5 (on page 22), getch(), halfdelay(), <curses.h>, XBD specification, Section 11.2,
  Parameters that Can be Set

CHANGE HISTORY
  First released in Issue 2.

Issue 4
  The entry is rewritten for clarity.
noecho()

NAME
  noecho — enable/disable terminal echo

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

DESCRIPTION
  Refer to echo().
NAME
   nonl — enable/disable newline translation

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

DESCRIPTION
   Refer to \textit{nl()}. 
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()` function 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.

EXAMPLES
None.

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 NOFLSH local mode flag specified in the XBD specification (`General Terminal Interface`).

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Section 3.5 (on page 22), `intrflush()`, `<curses.h>`, XBD specification, Section 11.2, Parameters that Can be Set (NOFLSH flag)

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

SYNOPSIS

```c
#include <curses.h>

int notimeout(WINDOW *win, bool bf);
void timeout(int delay);
void wtimeout(WINDOW *win, int delay);
```

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

The `timeout()` and `wtimeout()` functions set blocking or non-blocking read for the current or specified window based on the value of `delay`:
- `delay < 0` One or more blocking reads (indefinite waits for input) are used.
- `delay = 0` One or more non-blocking reads are used. Any Curses input function will fail if every character of the requested string is not immediately available.
- `delay > 0` Any Curses input function blocks for `delay` milliseconds and fails if there is still no input.

RETURN VALUE
Upon successful completion, the `notimeout()` function returns OK. Otherwise, it returns ERR.

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

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
- `Section 3.5` (on page 22), `getch()`, `halfdelay()`, `nodelay()`, `<curses.h>`, `XBD` specification, Section 11.2, 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
  These 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.

EXAMPLES
  None.

APPLICATION USAGE
  None.

RATIONALE
  None.

FUTURE DIRECTIONS
  None.

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.

  Issue 4, Version 2
    Corrections made to the SYNOPSIS.
NAME
pair_content, PAIR_NUMBER — get information on a color pair

SYNOPSIS
```
#include <curses.h>

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

DESCRIPTION
Refer to `can_change_color()`.
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.

EXAMPLES
None.

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.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
`echochar()`, `newpad()`, `<curses.h>`

CHANGE HISTORY
First released in Issue 4.

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

CURSES

pnoutrefresh()

NAME
pnoutrefresh, prefresh — refresh pads

SYNOPSIS
#include <curses.h>

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

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

DESCRIPTION
Refer to newpad().
NAME
printw — print formatted output in the current window

SYNOPSIS
#include <curses.h>
int printw(const char *fmt, ...);

DESCRIPTION
Refer to 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 `tiparm()` 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.

EXAMPLES
None.

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.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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

CHANGE HISTORY
First released in Issue 4.

Issue 4, Version 2
Corrections made to the SYNOPSIS.
NAME
putwin — dump window to a file

SYNOPSIS
#include <curses.h>

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

DESCRIPTION
Refer to getwin().
NAME
qiflush — enable queue flushing

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

DESCRIPTION
Refer to noqiflush().
raw()

NAME
   raw — set Raw Mode

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

DESCRIPTION
   Refer to cbreak().
NAME
redrawwin, wredrawln — line update status functions

SYNOPSIS
#include <curses.h>

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

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

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

ERRORS
No errors are defined.

EXAMPLES
None.

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.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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

CHANGE HISTORY
First released in Issue 4.
refresh() CURSES

NAME
refresh — refresh current window

SYNOPSIS

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

DESCRIPTION
Refer to doupdate().
NAME
reset_prog_mode, reset_shell_mode — restore program or shell terminal modes

SYNOPSIS
#include <curses.h>

int reset_prog_mode(void);
int reset_shell_mode(void);

DESCRIPTION
Refer to def_prog_mode().
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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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

```c
#include <term.h>

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

DESCRIPTION
Refer to `del_curterm()`.
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 initialization function cannot use the LINES and COLS external variables and cannot call wrefresh() or doupdate(), but may call wnoutrefresh().

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

RETURN VALUE
The ripoffline() function returns OK.

ERRORS
No errors are defined.

EXAMPLES
None.

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

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

RATIONALE
None.

FUTURE DIRECTIONS
None.

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

CHANGE HISTORY
First released in Issue 4.

Issue 4, Version 2
Corrections made to the SYNOPSIS.
NAME

savetty — save terminal mode

SYNOPSIS

#include <curses.h>

int savetty(void);

DESCRIPTION

Refer to resetty().
NAME
  scanw — convert formatted input from the current window

SYNOPSIS
  #include <curses.h>
  int scanw(const char *fmt, ...);

DESCRIPTION
  Refer to mvscanw().
NAME

scr_dump, scr_init, scr_restore, scr_set — screen file input/output functions

SYNOPSIS

```c
#include <curses.h>

int scr_dump(const char *filename);
int scr_init(const char *filename);
int scr_restore(const char *filename);
int scr_set(const char *filename);
```

DESCRIPTION

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

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

The `scr_init()` function reads the contents of the file named by `filename` and uses them to initialize the Curses data structures to what the terminal currently has on its screen. The next refresh operation bases any updates on this information, unless either of the following conditions is true:

- The terminal has been written to since the virtual screen was dumped to `filename`.
- The `terminfo` capabilities `rmcup` and `nrrmc` are defined for the current terminal.

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

RETURN VALUE

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

ERRORS

No errors are defined.

EXAMPLES

None.

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()`.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

delscreen(), doupdate(), 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.

Issue 4, Version 2
Corrections made to the SYNOPSIS.
NAME

scrl, scroll, wscrl — scroll a Curses window

SYNOPSIS

#include <curses.h>

EC
int scrl(int n);

int scroll(WINDOW *win);

EC
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 setscreg() is currently unspecified.

RETURN VALUE

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

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

In a future version, the interaction of these functions with setscreg() will be defined.

SEE ALSO

<curses.h>

CHANGE HISTORY

First released in Issue 4.

In previous versions, 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.
NAME
scrollok — enable or disable scrolling on a window

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

DESCRIPTION
Refer to clearok().
set_curterm( )

NAME
set_curterm — set current terminal

SYNOPSIS
#include <term.h>
TERMINAL *set_curterm(TERMINAL *nterm);

DESCRIPTION
Refer to del_curterm().
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, the `set_term()` function returns a pointer to the previous screen. Otherwise, it returns a null pointer.

ERRORS
No errors are defined.

EXAMPLES
None.

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

RATIONALE
None.

FUTURE DIRECTIONS
None.

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
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 initializes the object pointed to by wcval according to the character attributes in attrs, the color pair in color_pair, and the wide-character string pointed to by wch.

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

RETURN VALUE
Upon successful completion, the setcchar() function returns OK. Otherwise, it returns ERR.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Section 3.3 (on page 15), attroff(), can_change_color(), getcchar(), <curses.h>

CHANGE HISTORY
First released in Issue 4.

Issue 4, Version 2
Corrections made to the SYNOPSIS.
NAME
setscrreg — define software scrolling region

SYNOPSIS
#include <curses.h>
int setscrreg(int top, int bot);

DESCRIPTION
Refer to clearok().
NAME
setupterm — access the terminfo database

SYNOPSIS
#include <term.h>

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

DESCRIPTION
Refer to del_curterm().
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

```c
#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.
2  Align the end of `label` with the end of the space.

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

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

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

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

The `slk_attron()`, `slk_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 color.

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

**RETURN VALUE**

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

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

**ERRORS**

No errors are defined.

**EXAMPLES**

None.

**APPLICATION USAGE**

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

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

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

**RATIONALE**

None.

**FUTURE DIRECTIONS**

None.

**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 color 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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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 version.
NAME
start_color — initialize use of colors on terminal

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

DESCRIPTION
Refer to can_change_color().
NAME
stdscr — default window

SYNOPSIS

```c
#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()`.

RETURN VALUE
None.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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

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

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

DESCRIPTION
Refer to newpad().
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().
NAME
syncok, wcursyncup, wsyncdown, wsyncup — synchronize a window with its parents or children

SYNOPSIS

```c
#include <curses.h>

int syncok(WINDOW *win, bool bf);
void wcursyncup(WINDOW *win);
void wsyncdown(WINDOW *win);
void wsyncup(WINDOW *win);
```

DESCRIPTION
The `syncok()` function determines whether all ancestors of the specified window are implicitly touched whenever there is a change in the window. If `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 `wsyncup()` function unconditionally touches all ancestors of `win`.

RETURN VALUE
Upon successful completion, the `syncok()` function returns OK. Otherwise, it returns ERR.

The other functions do not return a value.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
Applications seldom call `wsyncdown()` because it is called by all refresh operations.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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

CHANGE HISTORY
First released in Issue 4.

Issue 4, Version 2
Corrections made to the SYNOPSIS.
NAME
termattrs — get supported terminal video attributes

SYNOPSIS
EC
#include <curses.h>

chtype termattrs(void);
attr_t term_attrs(void);

DESCRIPTION
The termattrs() function extracts the video attributes of the current terminal which is supported by the chtype 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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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

CHANGE HISTORY
First released in Issue 4.

Issue 4, Version 2
Corrections made to the SYNOPSIS; rewritten for clarity.
NAME
terminate — 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.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 7.1.1 (on page 338), `del_curterm()`, `getenv()` (in the XSH specification), `initscr()`, `<curses.h>`

CHANGE HISTORY

First released in Issue 4.
NAME

tigetflag, tigetnum, tigetstr, tiparm, tparm — retrieve or process capabilities from the terminfo database

SYNOPSIS

```c
#include <term.h>

int tigetflag(const char *capname);
int tigetnum(const char *capname);
char *tigetstr(const char *capname);
char *tiparm(const char *cap, ...);
char *tparm(const 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 7.1.3 (on page 340).

The `tiparm()` and `tparm()` functions take as `cap` a string capability. If `cap` is parameterized (as described in Section A.1.2, on page 354), these functions resolve the parameterization as described below.

If the parameterized string refers to one or more of the parameters `%p1` through `%p9`, then `tiparm()` fetches one argument for each `%pN` parameter, in order of `N` (that is, the first argument after `cap` is fetched for `%p1`, the second for `%p2`, and so on), and uses the corresponding argument value when pushing the `%pN` parameter on to the stack. The results are undefined if there are insufficient arguments for the parameterized string. If a `%pN` parameter is used in a string context (for example, if it is popped using `%l` or `%s`), the corresponding argument is fetched as type `char *`; otherwise, the argument is fetched as type `int`. If a `%pN` parameter is used more than once, at least one of the uses is in a string context, and the uses are not all in a string context, then the behavior is undefined. If parameter `%pN` is used and any of the lower numbered parameters, from `%p1` to `%p(N-1)`, are not used, the arguments corresponding to the unused parameters are fetched as type `int`.

If the parameterized string refers to one or more of the parameters `%p1` through `%p9`, then `tparm()` uses the values of `p1` through `p9`, respectively, when pushing the parameter on to the stack. If any of the parameters `%p1` through `%p9` is used in a string context (for example, if it is popped using `%l` or `%s`), the behavior is implementation-defined.

RETURN VALUE

Upon successful completion, the `tigetflag()`, `tigetnum()`, and `tigetstr()` functions 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, the `tiparm()` and `tparm()` functions return the capability pointed to by `cap` with parameterization resolved. Otherwise, they return a null pointer.

The return value from `tiparm()` and `tparm()` may point to static data which may be overwritten by a subsequent call to either function.
Curses Interfaces

ENHANCED CURSES

tigetflag()

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

For parameterized string capabilities, the application should pass the return value from tigetstr() to tiparm(), as described above. The tiparm() function is a replacement for the obsolescent tparm() function which cannot support string parameters on implementations where converting char * pointers to long int and back does not preserve their values.

Note that when converting old code to use tiparm() instead of tparm(), it is important to ensure that numeric argument values are passed to tiparm() as type int, or a type that promotes to int. With tparm(), numeric arguments could have any integer type and they would be converted to the correct type (long int) courtesy of the function prototype. However, tiparm() has a prototype that ends with an ellipsis, and therefore no such conversion is performed.

Applications intending to send terminal capabilities directly to the terminal (which should only be done using tputs() or putp()) instead of using Curses, normally should obey the following rules:

- Call reset_shell_mode() to restore the display modes before exiting.
- If using cursor addressing, output enter_ca_mode upon startup and output exit_ca_mode before exiting.
- If using shell escapes, output exit_ca_mode and call reset_shell_mode() before calling the shell; call reset_prog_mode() and output enter_ca_mode after returning from the shell.

All parameterized terminal capabilities defined in this document can be passed to tiparm(). All parameterized terminal capabilities defined in this document except pkey_key, pkey_local, pkey_plab, pkey_xmit, and plab_norm can be passed to tparm(). Some implementations (those where char * can be converted to long int and back without loss) might also allow pkey_key, pkey_local, pkey_plab, pkey_xmit, and plab_norm to 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 tiparm() and tparm() cannot parse these user-defined strings.

Applications should use the tigetflag(), tigetnum(), tigetstr(), and tparm() functions instead of the withdrawn tgetent(), tgetflag(), tgetnum(), tgetstr(), and tgoto() functions. Note that these replacement functions are only required to be supported on implementations supporting X/Open Enhanced Curses.

RATIONALE

The tiparm() function does not require that if parameter %pN is used in the parameterized string, %p1 through %p (N−1) must also be used. This is because some capabilities may have no use for some arguments in the definition for a specific terminal. An example is given in Section A.1.7 (on page 358) for sgr where the terminal has altscharset but does not have protect mode, and so the parameterized string would use %p9 but would not need to use %p8.

The arguments corresponding to unused parameters are fetched as type int, because numeric parameters are far more common than string parameters. If the need should arise for a string parameter to be (effectively) unused for a specific terminal, this can be handled by making the parameterized string push the parameter, pop it with %1, and then not use the length that was pushed by %1. This is sufficient for tiparm() to see the parameter being used in a string context, so that it will still expect the corresponding argument to have type char *.
FUTURE DIRECTIONS
None.

SEE ALSO
def_prog_mode(), putp(), <term.h>

CHANGE HISTORY
First released in Issue 4.

Issue 7
The prototypes for the tigetflag(), tigetnum(), and tigetstr() functions are updated.
NAME
timeout — control blocking on input

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

DESCRIPTION
Refer to notimeout().
NAME

tiparm — format terminfo string capability

SYNOPSIS

```c
#include <term.h>

char *tiparm(const char *cap, ...);
```

DESCRIPTION

Refer to tigetflag().
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().
NAME
tparm — format terminfo string capability

SYNOPSIS
#include <term.h>
char *tparm(const char *cap, long p1, long p2, long p3, long p4,
            long p5, long p6, long p7, long p8, long p9);

DESCRIPTION
Refer to tigetflag().
NAME

tputs — output commands to the terminal

SYNOPSIS

```c
#include <curses.h>

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

DESCRIPTION

Refer to `putp()`.
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, the typeahead() function returns OK. Otherwise, it returns ERR.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Section 3.5 (on page 22), doupdate(), getch(), initscr(), <curses.h>, XBD specification, Section 11.2, 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 versions.
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, the unctrl() function returns the generated string. Otherwise, it returns a null pointer.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

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 versions.
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.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Section 3.5 (on page 22), `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().
**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.

**EXAMPLES**

None.

**APPLICATION USAGE**

None.

**RATIONALE**

None.

**FUTURE DIRECTIONS**

None.

**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 color pair number and a `void *`, and thus support the attribute constants with the WA_prefix.

The `opts` argument is reserved for definition in a future version. 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.

EXAMPLES

None.

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.

RATIONALE

None.
FUTURE DIRECTIONS
None.

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 color handling functions.
NAME
vline — draw vertical line

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

DESCRIPTION
Refer to hline().
vline_set()  

NAME
vline_set — draw vertical line from complex character and rendition

SYNOPSIS

```c
#include <curses.h>
int vline_set(const cchar_t *ch, int n);
```

DESCRIPTION
Refer to hline_set().
NAME
vw_printw — print formatted output in window

SYNOPSIS

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

int vw_printw(WINDOW *, const 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, the `vw_printw`() function returns OK. Otherwise, it returns ERR.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Applications should use the `vw_printw`() function instead of the withdrawn `vwprintw`() function. Note that this replacement function is only required to be supported on implementations supporting X/Open Enhanced Curses.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

`mvprintw()`, `fprintf()` (in the XSH specification), `<curses.h>`, `<stdarg.h>` (in the XBD specification)

CHANGE HISTORY

First released in Issue 4.

Issue 4, Version 2

Corrections made to the SYNOPSIS.

Issue 7

The prototype for the `vw_printw`() function is updated.
NAME
vw_scanw — convert formatted input from a window

SYNOPSIS

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

int vw_scanw(WINDOW *, const char *, va_list varglist);
```

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, the `vw_scanw()` function returns OK. Otherwise, it returns ERR.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
Applications should use the `vw_scanw()` function instead of the withdrawn `vwscanf()` function. Note that this replacement function is only required to be supported on implementations supporting X/Open Enhanced Curses.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
`fscanf()` (in the XSH specification), `mvscanf()`, `<curses.h>`, `<stdarg.h>` (in the XBD specification)

CHANGE HISTORY
First released in Issue 4.

Issue 4, Version 2
Corrections made to the SYNOPSIS and APPLICATION USAGE.

Issue 7
The prototype for the `vw_scanw()` function is updated.
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.

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

(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 as follows.

Note: The asterisk (*) denotes that there is no corresponding function without the w prefix.

<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>addchstr()</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>waddnwstr()</td>
<td>addnwstr()</td>
</tr>
<tr>
<td>waddwstr()</td>
<td>addwstr()</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>wattton()</td>
<td>attron()</td>
</tr>
<tr>
<td>wattset()</td>
<td>attrset()</td>
</tr>
<tr>
<td>wattr_get()</td>
<td>attr_get()</td>
</tr>
<tr>
<td>wattr_off()</td>
<td>attr_get()</td>
</tr>
<tr>
<td>wattr_on()</td>
<td>attr_get()</td>
</tr>
<tr>
<td>wattr_set()</td>
<td>attr_get()</td>
</tr>
<tr>
<td>wbkgd()</td>
<td>bkgd()</td>
</tr>
<tr>
<td>wbkgdset()</td>
<td>bkgd()</td>
</tr>
<tr>
<td>wbkgrnd()</td>
<td>bkgrnd()</td>
</tr>
<tr>
<td>wbkgrndset()</td>
<td>bkgrnd()</td>
</tr>
<tr>
<td>wborder()</td>
<td>border()</td>
</tr>
<tr>
<td>wborder_set()</td>
<td>border_set()</td>
</tr>
<tr>
<td>wchgat()</td>
<td>chgat()</td>
</tr>
<tr>
<td>wclear()</td>
<td>clear()</td>
</tr>
<tr>
<td>wcrltobot()</td>
<td>crlrtobot()</td>
</tr>
<tr>
<td>wcrltoool()</td>
<td>crltoool()</td>
</tr>
<tr>
<td>wcursorsty()</td>
<td>syncof()</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>
<tr>
<td>werase()</td>
<td>clear()</td>
</tr>
<tr>
<td>wgetbkgrnd()</td>
<td>bkgrnd()</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>Function</td>
<td>Refer to</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------</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>winchstr()</td>
<td>inchnstr()</td>
</tr>
<tr>
<td>winnstr()</td>
<td>innstr()</td>
</tr>
<tr>
<td>winnwstr()</td>
<td>inwwstr()</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>winsstr()</td>
<td>insnstr()</td>
</tr>
<tr>
<td>winnstr()</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>inwwstr()</td>
</tr>
<tr>
<td>win_wch()</td>
<td>in_wch()</td>
</tr>
<tr>
<td>win_wchstr()</td>
<td>in_wchstr()</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>wredrawwin()</td>
<td>redrawwin()</td>
</tr>
<tr>
<td>wrefresh()</td>
<td>doupdate()</td>
</tr>
<tr>
<td>wscanw()</td>
<td>mvscanw()</td>
</tr>
<tr>
<td>wscrl()</td>
<td>scrl()</td>
</tr>
<tr>
<td>wsetscreq()</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>wtouchin(*)</td>
<td>is_linetouched()</td>
</tr>
<tr>
<td>wline()</td>
<td>hline()</td>
</tr>
<tr>
<td>wline_set()</td>
<td>hline_set()</td>
</tr>
</tbody>
</table>
NAME
wadd_wch — add a complex character and rendition to a window

SYNOPSIS
#include <curses.h>

int wadd_wch(WINDOW *win, const cchar_t *wch);

DESCRIPTION
Refer to add_wch().
NAME
wadd_wchnstr, wadd_wchstr — add an array of complex characters and renditions to a window

SYNOPSIS

#include <curses.h>

int wadd_wchnstr(WINDOW *win, const cchar_t *wchstr, int n);
int wadd_wchstr(WINDOW *win, const cchar_t *wchstr);

DESCRIPTION
Refer to add_wchnstr().
NAME
waddch — add a single-byte character and rendition to a window and advance the cursor

SYNOPSIS
#include <curses.h>
int waddch(WINDOW *win, const chtype ch);

DESCRIPTION
Refer to addch().
NAME
waddchstr, waddchnstr — add string of single-byte characters and renditions to a window

SYNOPSIS
#include <curses.h>

int waddchstr(WINDOW *win, const chtype *chstr);

int waddchnstr(WINDOW *win, const chtype *chstr, int n);

DESCRIPTION
Refer to addchstr().
NAME
waddnstr, waddstr — add a string of multi-byte characters without rendition to a window and advance cursor

SYNOPSIS

```c
#include <curses.h>

int waddnstr(WINDOW *win, const char *str, int n);
int waddstr(WINDOW *win, const char *str);
```

DESCRIPTION
Refer to `addnstr()`.
waddnwstr()  ENHANCED CURSES  Curses Interfaces

NAME
waddnwstr, waddwstr — add a wide-character string to a window and advance the cursor

SYNOPSIS

#include <curses.h>

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

DESCRIPTION
Refer to addnwstr().
NAME
wattr_get, wattr_off, wattr_on, wattr_set — window attribute control functions

SYNOPSIS

```c
#include <curses.h>

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);
```

DESCRIPTION
Refer to `attr_get()`.
NAME
  wattroff, wattron, wattrset — restricted window attribute control functions

SYNOPSIS

```c
#include <curses.h>

int wattroff(WINDOW *win, int attrs);
int wattron(WINDOW *win, int attrs);
int wattrset(WINDOW *win, int attrs);
```

DESCRIPTION

  Refer to `attroff()`.
NAME
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
#include <curses.h>

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

DESCRIPTION
Refer to bkgd().
NAME

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

DESCRIPTION

Refer to `bkgrnd()`. 
NAME
wborder — draw borders from single-byte characters and renditions

SYNOPSIS

```c
#include <curses.h>

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

DESCRIPTION

Refer to `border()`.
NAME

wborder_set — draw borders from complex characters and renditions

SYNOPSIS

```c
#include <curses.h>

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

Refer to `border_set()`.
NAME
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);
```

DESCRIPTION

Refer to `chgat()`.
NAME
wclear, werase — clear a window

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

DESCRIPTION
Refer to clear().
NAME
   wclrtobot — clear from cursor to end of window

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

DESCRIPTION
   Refer to clrtobot().
**NAME**

wclrtoeol — clear from cursor to end of line

**SYNOPSIS**

```
#include <curses.h>
int wclrtoeol(WINDOW *win);
```

**DESCRIPTION**

Refer to `clrtoeol()`. 
NAME
wcolor_set — window attribute control functions

SYNOPSIS
#include <curses.h>

int wcolor_set(WINDOW *win, short color_pair_number, void *opts);

DESCRIPTION
Refer to attr_get().
NAME
wcursyncup — synchronize a window with its parents or children

SYNOPSIS

```c
#include <curses.h>
void wcursyncup(WINDOW *win);
```

DESCRIPTION

Refer to `syncok()`.
NAME
  wdelch — delete a character from a window

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

DESCRIPTION
  Refer to \texttt{delch()}. 
NAME
wdeleteln — delete lines in a window

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

DESCRIPTION
Refer to deleteln().
NAME
wecho_wchar — write a complex character and immediately refresh the window

SYNOPSIS
#include <curses.h>

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

DESCRIPTION
Refer to echo_wchar().
NAME
wechochar — echo single-byte character and rendition to a window and refresh

SYNOPSIS
#include <curses.h>

int wechochar(WINDOW *win, const chtype ch);

DESCRIPTION
Refer to echochar().
NAME
wget_wch — get a wide character from a terminal

SYNOPSIS

```c
#include <curses.h>

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

DESCRIPTION

Refer to `get_wch()`. 
NAME
wgetch — get a single-byte character from the terminal

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

DESCRIPTION
Refer to \textit{getch()}. 
NAME
wgetn_wstr, wget_wstr — get an array of wide characters and function key codes from a terminal

SYNOPSIS
#include <curses.h>

int wgetn_wstr(WINDOW *win, wint_t *wstr, int n);
int wget_wstr(WINDOW *win, wint_t *wstr);

DESCRIPTION
Refer to getn_wstr().
NAME
wgetnstr, wgetstr — get a multi-byte character string from the terminal

SYNOPSIS
#include <curses.h>

int wgetnstr(WINDOW *win, char *str, int n);
int wgetstr(WINDOW *win, char *str);

DESCRIPTION
Refer to getnstr().
NAME
whline, wvline — draw lines from single-byte characters and renditions

SYNOPSIS

#include <curses.h>

int whline(WINDOW *win, chtype ch, int n);
int wvline(WINDOW *win, chtype ch, int n);

DESCRIPTION
Refer to hline().
NAME
whline_set, wvline_set — draw lines from complex characters and renditions

SYNOPSIS
#include <curses.h>

int whline_set(WINDOW *win, const cchar_t *wch, int n);
int wvline_set(WINDOW *win, const cchar_t *wch, int n);

DESCRIPTION
Refer to hline_set().
NAME

win_wch — extract a complex character and rendition from a window

SYNOPSIS

```c
#include <curses.h>

int win_wch(WINDOW *win, cchar_t *wcval);
```

DESCRIPTION

Refer to `in_wch`.
NAME

win_wchnstr, win_wchstr — extract an array of complex characters and renditions from a window

SYNOPSIS

```c
#include <curses.h>

int win_wchnstr(WINDOW *win, cchar_t *wchstr, int n);
int win_wchstr(WINDOW *win, cchar_t *wchstr);
```

DESCRIPTION

Refer to `in_wchnstr()`. 
NAME
   winch — input a single-byte character and rendition from a window

SYNOPSIS
   #include <curses.h>
   chtype winch(WINDOW *win);

DESCRIPTION
   Refer to inch().
NAME

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

SYNOPSIS

```
#include <curses.h>

int winchnstr(WINDOW *win, chtype *chstr, int n);
int winchstr(WINDOW *win, chtype *chstr);
```

DESCRIPTION

Refer to `inchnstr()`.
NAME
winnstr, winstr — input a multi-byte character string from a window

SYNOPSIS

```c
#include <curses.h>

int winnstr(WINDOW *win, char *str, int n);
int winstr(WINDOW *win, char *str);
```

DESCRIPTION
Refer to `innstr()`.
winnwstr()  

NAME
winnwstr, winwstr — input a string of wide characters from a window

SYNOPSIS

```
#include <curses.h>

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

DESCRIPTION

Refer to innwstr().
NAME
wins_nwstr, wins_wstr — insert a wide-character string into a window

SYNOPSIS

```
#include <curses.h>

int wins_nwstr(WINDOW *win, const wchar_t *wstr, int n);
int wins_wstr(WINDOW *win, const wchar_t *wstr);
```

DESCRIPTION

Refer to `ins_nwstr()`.
NAME
wins_wch — insert a complex character and rendition into a window

SYNOPSIS

```c
#include <curses.h>

int wins_wch(WINDOW *win, const cchar_t *wch);
```

DESCRIPTION
Refer to `ins_wch()`.
NAME
winsch — insert a single-byte character and rendition into a window

SYNOPSIS
#include <curses.h>
int winsch(WINDOW *win, chtype ch);

DESCRIPTION
Refer to insch().
NAME
winsdelln — delete or insert lines into a window

SYNOPSIS
```
#include <curses.h>

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

DESCRIPTION
Refer to insdelln().
NAME
  winsertln — insert lines into a window

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

DESCRIPTION
  Refer to insertln().
winsnstr()  ENHANCED CURSES  Curses Interfaces

NAME
winsnstr, winsstr — insert a multi-byte character string into a window

SYNOPSIS

```c
#include <curses.h>

int winsnstr(WINDOW *win, const char *str, int n);
int winsstr(WINDOW *win, const char *str);
```

DESCRIPTION
Refer to insnstr().
NAME
wmove — window cursor location functions

SYNOPSIS

#include <curses.h>

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

DESCRIPTION
Refer to move().
WNOUTREFRESH( )

NAME
wnoutrefresh, wrefresh — refresh windows and lines

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

DESCRIPTION
Refer to doupdate().
NAME
   wprintw — print formatted output in window

SYNOPSIS
   #include <curses.h>
   int wprintw(WINDOW *win, const char *fmt, ...);

DESCRIPTION
   Refer to mvprintw().
NAME
wredrawln — line update status functions

SYNOPSIS
#include <curses.h>

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

DESCRIPTION
Refer to redrawwin().
NAME
wscanw — convert formatted input from a window

SYNOPSIS
#include <curses.h>

int wscanw(WINDOW *win, const char *fmt, ...);

DESCRIPTION
Refer to mvscanw().
NAME

wscrl — scroll a Curses window

SYNOPSIS

```c
#include <curses.h>
int wscrl(WINDOW *win, int n);
```

DESCRIPTION

Refer to `scrl()`.
NAME
  wsetscrreg — terminal output control functions

SYNOPSIS
  #include <curses.h>
  int wsetscrreg(WINDOW *win, int top, int bot);

DESCRIPTION
  Refer to clearok().
wstandend()  CURSES  Curses Interfaces

NAME
   wstandend, wstandout — set and clear window attributes

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

DESCRIPTION
   Refer to standend().
NAME
wsyncdown, wsyncup — synchronize a window with its parents or children

SYNOPSIS
#include <curses.h>
void wsyncdown(WINDOW *win);
void wsyncup(WINDOW *win);

DESCRIPTION
Refer to syncok().
wtimeout()  

ENHANCED CURSES

Curses Interfaces

NAME

wtimeout — control blocking on input

SYNOPSIS

```c
#include <curses.h>

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

DESCRIPTION

Refer to `notimeout()`.
NAME
wtouchln — window refresh control functions

SYNOPSIS

```c
#include <curses.h>

int wtouchln(WINDOW *win, int y, int n, int changed);
```

DESCRIPTION
Refer to `is_linetouched()`.
NAME
wunctrl — generate printable representation of a wide character

SYNOPSIS
```
#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, the wunctrl() function returns the generated string. Otherwise, it returns a null pointer.

ERRORS
No errors are defined.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
keyname(), unctrl(), <curses.h>

CHANGE HISTORY
First released in Issue 4.
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 optimization functions

SYNOPSIS
#include <curses.h>

DESCRIPTION

Objects
The <curses.h> header provides a declaration for COLOR_PAIRS, COLORS, COLS, curscr,
LINES, and stdscr.

Macros
The following macros are defined:
EOF Function return value for end-of-file, as described in <stdio.h>.
ERR Function return value for failure.
FALSE Boolean false value.
KEY_CODE_YES Function return value indicating that a wint_t variable contains a key
code.
OK Function return value for success.
TRUE Boolean true value.
WEOF Wide-character function return value for end-of-file, as described in <wchar.h>.

The following macro is defined:
_XOPEN_CURSES X/Open Enhanced Curses test macro.

Data Types
The following data type is defined as a macro:
bool As described in <stdbool.h>.

The following data types are defined through typedef:
attr_t An OR’ed set of attributes.
chtype A character, attributes, and a color-pair.
SCREEN An opaque terminal representation.
wchar_t As described in <stddef.h>.
wint_t As described in <wchar.h>.
cchar_t References a string of wide characters.
WINDOW An opaque window representation.

These data types are described in more detail in Section 2.3 (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 macros are used to manipulate objects of type `attr_t`:

<table>
<thead>
<tr>
<th>Macro</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 macros are used to manipulate attribute bits in objects of type `chtype`:

<table>
<thead>
<tr>
<th>Macro</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>
<tr>
<td>A_UNDERLINE</td>
<td>Underlining</td>
</tr>
</tbody>
</table>

These attribute flags need not be distinct except when `_XOPEN_CURSES` is defined.

The following macros can be used as bit-masks to extract the components of a `chtype`:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_ATTRIBUTES</td>
<td>Bit-mask to extract attributes</td>
</tr>
<tr>
<td>A_CHARTEXT</td>
<td>Bit-mask to extract a character</td>
</tr>
<tr>
<td>A_COLOR</td>
<td>Bit-mask to extract color-pair information</td>
</tr>
</tbody>
</table>
Line-Drawing Macros

The `<curses.h>` header defines the macros shown in the leftmost two columns of the following table for use in drawing lines. The macros that begin with ACS_ are `char` constants. The macros that begin with WACS_ are `cchar_t` constants used 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 `acs` capability as described in Section A.1.12 (on page 361).

<table>
<thead>
<tr>
<th>char Constant</th>
<th>cchar_t Constant</th>
<th>POSIX Locale Default</th>
<th>Glyph Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACS_ULCORNER</td>
<td>WACS_ULCORNER</td>
<td>+</td>
<td>upper left-hand corner</td>
</tr>
<tr>
<td>ACS_LLCORNER</td>
<td>WACS_LLCORNER</td>
<td>+</td>
<td>lower left-hand corner</td>
</tr>
<tr>
<td>ACS_URCORNER</td>
<td>WACS_URCORNER</td>
<td>+</td>
<td>upper right-hand corner</td>
</tr>
<tr>
<td>ACS_LRCORNER</td>
<td>WACS_LRCORNER</td>
<td>+</td>
<td>lower right-hand corner</td>
</tr>
<tr>
<td>ACS_RTEE</td>
<td>WACS_RTEE</td>
<td>+</td>
<td>right tee (−)</td>
</tr>
<tr>
<td>ACS_LTEE</td>
<td>WACS_LTEE</td>
<td>+</td>
<td>left tee (│)</td>
</tr>
<tr>
<td>ACS_BTTEE</td>
<td>WACS_BTTEE</td>
<td>+</td>
<td>bottom tee (│)</td>
</tr>
<tr>
<td>ACS_TTEE</td>
<td>WACS_TTEE</td>
<td>+</td>
<td>top tee (T)</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>
Color-Related Macros

The following color-related macros are defined:

COLOR_BLACK
COLOR_BLUE
COLOR_GREEN
COLOR_CYAN
COLOR_RED
COLOR_MAGENTA
COLOR_YELLOW
COLOR_WHITE

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

```c
int COLOR_PAIR(int);
int PAIR_NUMBER(int);
```

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 macros representing function key values are defined and have distinct values where each value is less than \( \text{CHAR_MIN} \) or greater than \( \text{UCHAR_MAX} \).

<table>
<thead>
<tr>
<th>Key Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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_BACKSPACE</td>
<td>Backspace</td>
</tr>
<tr>
<td>KEY_BEG</td>
<td>Beginning key</td>
</tr>
<tr>
<td>KEY_BREAK</td>
<td>Break key</td>
</tr>
<tr>
<td>KEY_BTAB</td>
<td>Back tab key</td>
</tr>
<tr>
<td>KEY_C1</td>
<td>Lower left of keypad</td>
</tr>
<tr>
<td>KEY_C3</td>
<td>Lower right of keypad</td>
</tr>
<tr>
<td>KEY_CANCEL</td>
<td>Cancel key</td>
</tr>
<tr>
<td>KEY_CATAB</td>
<td>Clear all tabs</td>
</tr>
<tr>
<td>KEY_CLEAR</td>
<td>Clear screen</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_CTAB</td>
<td>Clear tab</td>
</tr>
<tr>
<td>Key Code</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>KEY_DC</td>
<td>Delete character</td>
</tr>
<tr>
<td>KEY_DL</td>
<td>Delete line</td>
</tr>
<tr>
<td>KEY_DOWN</td>
<td>Down arrow key</td>
</tr>
<tr>
<td>KEY_EIC</td>
<td>Exit insert char mode</td>
</tr>
<tr>
<td>KEY_END</td>
<td>End key</td>
</tr>
<tr>
<td>KEY_ENTER</td>
<td>Enter or send</td>
</tr>
<tr>
<td>KEY_EOL</td>
<td>Clear to end of line</td>
</tr>
<tr>
<td>KEY_EOS</td>
<td>Clear to end of screen</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_FIND</td>
<td>Find key</td>
</tr>
<tr>
<td>KEY_HELP</td>
<td>Help key</td>
</tr>
<tr>
<td>KEY_HOME</td>
<td>Home key</td>
</tr>
<tr>
<td>KEY_IC</td>
<td>Insert char or enter insert mode</td>
</tr>
<tr>
<td>KEY_IL</td>
<td>Insert line</td>
</tr>
<tr>
<td>KEY_LEFT</td>
<td>Left arrow key</td>
</tr>
<tr>
<td>KEY_LL</td>
<td>Home down or bottom</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_NPAGE</td>
<td>Next page</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_PPAGE</td>
<td>Previous page</td>
</tr>
<tr>
<td>KEY_PREVIOUS</td>
<td>Previous object key</td>
</tr>
<tr>
<td>KEY_PRINT</td>
<td>Print or copy</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_RESET</td>
<td>Reset or hard reset</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_RIGHT</td>
<td>Right arrow 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_SSCAN</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_SHCREATE</td>
<td>Shifted create key</td>
</tr>
<tr>
<td>KEY_SDC</td>
<td>Shifted delete char key</td>
</tr>
<tr>
<td>KEY_SSL</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_SF</td>
<td>Scroll 1 line forward</td>
</tr>
</tbody>
</table>
## Key Code

<table>
<thead>
<tr>
<th>Key Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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_SPREVIOUS</td>
<td>Shifted prev key</td>
</tr>
<tr>
<td>KEY_SPRINT</td>
<td>Shifted print key</td>
</tr>
<tr>
<td>KEY_SR</td>
<td>Scroll 1 line backward (reverse)</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_SRESET</td>
<td>Soft (partial) reset</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_STAB</td>
<td>Set tab</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>
<tr>
<td>KEY_UP</td>
<td>Up arrow key</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>A1</th>
<th>UP</th>
<th>A3</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT</td>
<td>B2</td>
<td>RIGHT</td>
</tr>
<tr>
<td>C1</td>
<td>DOWN</td>
<td>C3</td>
</tr>
</tbody>
</table>

Each legend, such as A1, corresponds to a macro for a key code from the preceding table, such as KEY_A1.

## 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 addnstr(const char *, int);
int addnwstr(const wchar_t *, 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);
```
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int attr_get(attr_t *, short *, void *)</code></td>
<td>Get an attribute value</td>
</tr>
<tr>
<td><code>int attr_off(attr_t, void *)</code></td>
<td>Turn off an attribute</td>
</tr>
<tr>
<td><code>int attr_on(attr_t, void *)</code></td>
<td>Turn on an attribute</td>
</tr>
<tr>
<td><code>int attr_set(attr_t, short, void *)</code></td>
<td>Set an attribute value</td>
</tr>
<tr>
<td><code>int baudrate(void)</code></td>
<td>Get the baud rate</td>
</tr>
<tr>
<td><code>int beep(void)</code></td>
<td>Make a beep sound</td>
</tr>
<tr>
<td><code>int bkgd(chtype)</code></td>
<td>Set the background color</td>
</tr>
<tr>
<td><code>void bkgdset(chtype)</code></td>
<td>Set the background color set</td>
</tr>
<tr>
<td><code>int bkgrnd(const cchar_t *)</code></td>
<td>Get the background color</td>
</tr>
<tr>
<td><code>void bkgrndset(const cchar_t *)</code></td>
<td>Set the background color set</td>
</tr>
<tr>
<td><code>int border(chtype, chtype, chtype, chtype, chtype)</code></td>
<td>Set a border</td>
</tr>
<tr>
<td><code>int border_set(const cchar_t *, const cchar_t *, const cchar_t *, const cchar_t *, const cchar_t *)</code></td>
<td>Set a border</td>
</tr>
<tr>
<td><code>int box(WINDOW *, chtype, chtype)</code></td>
<td>Draw a box</td>
</tr>
<tr>
<td><code>int box_set(WINDOW *, const cchar_t *, const cchar_t *)</code></td>
<td>Set a box</td>
</tr>
<tr>
<td><code>bool can_change_color(void)</code></td>
<td>Check if color changes are supported</td>
</tr>
<tr>
<td><code>int cbreak(void)</code></td>
<td>Enable break characters</td>
</tr>
<tr>
<td><code>int chgat(int, attr_t, short, const void *)</code></td>
<td>Change attribute</td>
</tr>
<tr>
<td><code>int clearok(WINDOW *, bool)</code></td>
<td>Enable clearing of output</td>
</tr>
<tr>
<td><code>int clear(void)</code></td>
<td>Clear the screen</td>
</tr>
<tr>
<td><code>int clrtoeol(void)</code></td>
<td>Clear to end of line</td>
</tr>
<tr>
<td><code>int color_content(short, short *, short *, short *)</code></td>
<td>Color content</td>
</tr>
<tr>
<td><code>int color_set(short, short *)</code></td>
<td>Set color content</td>
</tr>
<tr>
<td><code>int copywin(const WINDOW *, WINDOW *, int, int, int, int, int, int)</code></td>
<td>Copy window</td>
</tr>
<tr>
<td><code>int curs_set(int)</code></td>
<td>Enable cursor display</td>
</tr>
<tr>
<td><code>int def_prog_mode(void)</code></td>
<td>Set up program mode</td>
</tr>
<tr>
<td><code>int def_shell_mode(void)</code></td>
<td>Set up shell mode</td>
</tr>
<tr>
<td><code>int delay_output(int)</code></td>
<td>Delay output</td>
</tr>
<tr>
<td><code>int delch(void)</code></td>
<td>Delete character</td>
</tr>
<tr>
<td><code>int deleteln(void)</code></td>
<td>Delete line of text</td>
</tr>
<tr>
<td><code>void delscreen(SCREEN *)</code></td>
<td>Delete the screen</td>
</tr>
<tr>
<td><code>int delwin(WINDOW *)</code></td>
<td>Delete window</td>
</tr>
<tr>
<td><code>WINDOW *derwin(WINDOW *, int, int, int, int)</code></td>
<td>Derive window</td>
</tr>
<tr>
<td><code>int doupdate(void)</code></td>
<td>Update display</td>
</tr>
<tr>
<td><code>WINDOW *dupwin(WINDOW *)</code></td>
<td>Duplicate window</td>
</tr>
<tr>
<td><code>int echo(void)</code></td>
<td>Echo text</td>
</tr>
<tr>
<td><code>int echochar(const chtype)</code></td>
<td>Echo a character</td>
</tr>
<tr>
<td><code>int echo_wchar(const cchar_t *)</code></td>
<td>Echo a wide character</td>
</tr>
<tr>
<td><code>int endwin(void)</code></td>
<td>End terminal display</td>
</tr>
<tr>
<td><code>char erasechar(void)</code></td>
<td>Erase character</td>
</tr>
<tr>
<td><code>int erase(void)</code></td>
<td>Erase the screen</td>
</tr>
<tr>
<td><code>int erasewchar(wchar_t *)</code></td>
<td>Erase a wide character</td>
</tr>
<tr>
<td><code>void filter(void)</code></td>
<td>Filter input</td>
</tr>
<tr>
<td><code>int flash(void)</code></td>
<td>Flash the display</td>
</tr>
<tr>
<td><code>int flushinp(void)</code></td>
<td>Flush input</td>
</tr>
<tr>
<td><code>chttype getbkgd(WINDOW *)</code></td>
<td>Get the background color</td>
</tr>
<tr>
<td><code>int getbkgrnd(cchar_t *)</code></td>
<td>Get the background color set</td>
</tr>
<tr>
<td><code>int getcchar(const cchar_t *, wchar_t *, attr_t *, short *)</code></td>
<td>Get a character</td>
</tr>
</tbody>
</table>
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);

chtype
inch(void);

int
inchnstr(chtype *, int);

int
inchstr(chtype *);

WINDOW *
initcolor(short, short, short, short);

int
init_pair(short, short, short);

int
innstr(char *, int);

int
innwstr(wchar_t *, int);

int
insch(chtype);

int
insdelln(int);

int
insertln(void);

int
insnstr(const char *, int);

int
ins_nwstr(const wchar_t *, int);

int
insstr(const char *);

int
instr(char *);

int
ins_wch(const cchar_t *);

int
ins_wstr(const wchar_t *);

int
intrflush(WINDOW *, bool);

int
in_wch(cchar_t *);

int
in_wchnstr(cchar_t *, int);

int
in_wchstr(cchar_t *);

int
inwstr(wchar_t *);

bool
isendwin(void);

bool
is_linetouched(WINDOW *, int);

bool
is_wintouched(WINDOW *);

char *
keyname(int);

char *
key_name(wchar_t);

int
keypad(WINDOW *, bool);

char
killchar(void);

int
killwchar(wchar_t *);

int
leaveok(WINDOW *, bool);

char
*longname(void);

int
meta(WINDOW *, bool);

int
move(int, int);

int
mvaddch(int, int, const chtype);
<curses.h>

CURSES

Headers

- `int mvaddchnstr(int, int, const chtype *, int);`
- `int mvaddchstr(int, int, const chtype *);`
- `int mvaddnstr(int, int, const char *, int);`
- `int mvaddnwstr(int, int, const wchar_t *, int);`
- `int mvaddstr(int, int, const char *);`
- `int mvadd_wch(int, int, const cchar_t *);`
- `int mvadd_wchnstr(int, int, const cchar_t *, int);`
- `int mvadd_wchstr(int, int, const cchar_t *);`
- `int mvaddwstr(int, int, const wchar_t *);`
- `int mvchgat(int, int, int, attr_t, short, const void *);`
- `int mvcur(int, int, int, int);`
- `int mvdelch(int, int);`
- `int mvderwin(WINDOW *, int, int, int);`
- `int mvgetch(int, int);`
- `int mvgetnstr(int, int, char *, int);`
- `int mvgetn_wstr(int, int, wint_t *, int);`
- `int mvgetstr(int, int, char *);`
- `int mvget_wch(int, int, wint_t *);`
- `int mvget_wstr(int, int, wint_t *);`
- `int mvhline(int, int, chtype, int);`
- `int mvhline_set(int, int, const cchar_t *, int);`
- `chtype mvinch(int, int);`
- `int mvchinchnstr(int, int, chtype *, int);`
- `int mvchstr(int, int, chtype *);`
- `int mvinnstr(int, int, char *, int);`
- `int mvinnwstr(int, int, wchar_t *, int);`
- `int mvinsch(int, int, chtype);`
- `int mvinsnstr(int, int, const char *, int);`
- `int mvins_nwstr(int, int, const wchar_t *, int);`
- `int mvinsstr(int, int, const char *);`
- `int mvirs(int, int, char *);`
- `int mvins_wch(int, int, const cchar_t *);`
- `int mvins_wstr(int, int, const wchar_t *);`
- `int mvins_wch(int, int, cchar_t *);`
- `int mvins_wchnstr(int, int, cchar_t *, int);`
- `int mvins_wchstr(int, int, cchar_t *);`
- `int mvinswstr(int, int, wchar_t *);`
- `int mvprintw(int, int, const char *, ...);`
- `int mvwchar(int, int, const char *, ...);`
- `chtype mvch(int, int, int);`
- `int mvch_set(int, int, const cchar_t *, int);`
- `int mvwaddch(WINDOW *, int, int, chtype *);`
- `int mvwaddchnstr(WINDOW *, int, int, const chtype *, int);`
- `int mvwaddchstr(WINDOW *, int, int, const chtype *);`
- `int mvwaddnstr(WINDOW *, int, int, const char *, int);`
- `int mvwaddnwstr(WINDOW *, int, int, const wchar_t *, int);`
- `int mvwaddstr(WINDOW *, int, int, const char *);`
- `int mvwadd_wch(WINDOW *, int, int, const cchar_t *);`
- `int mvwadd_wchnstr(WINDOW *, int, int, const cchar_t *, int);`
- `int mvwadd_wchstr(WINDOW *, int, int, const cchar_t *);`
- `int mvwaddwstr(WINDOW *, int, int, const wchar_t *);`
- `int mvwchgat(WINDOW *, int, int, int, attr_t, short, const void *);`

314 Technical Standard 2009
const void *);

int mvdelselch(WINDOW *, int, int);

int mvwgetch(WINDOW *, int, int);

EC int mvwgetnstr(WINDOW *, int, int, char *, int);
int mvwgetn_wstr(WINDOW *, int, int, wint_t *, int);

EC int mvwgetch(WINDOW *, int, int, wint_t *);
int mvwgetstr(WINDOW *, int, int, chtype, int);
int mvwhline_set(WINDOW *, int, int, const cchar_t *, int);

int mvwin(WINDOW *, int, int);

chtype mvwinch(WINDOW *, int, int);

EC int mvwinchnstr(WINDOW *, int, int, chtype *, int);

int mvwvinchstr(WINDOW *, int, int, chtype *);
int mvwinnstr(WINDOW *, int, int, char *, int);

int mvwinnwstr(WINDOW *, int, int, wchar_t *, int);

int mvwinsch(WINDOW *, int, int, chtype);

EC int mvwinsnstr(WINDOW *, int, int, const char *, int);

int mvwinsnstr(WINDOW *, int, int, const char *, int);
int mvwinnwstr(WINDOW *, int, int, wchar_t *, int);

int mvwprintw(WINDOW *, int, int, const char *, ...);

int mvwscanw(WINDOW *, int, int, const char *, ...);

WINDOW *newpad(int, int);

SCREEN *newterm(const 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);

EC void noqiflush(void);

int noraw(void);

EC int notimeout(WINDOW *, bool);

int overlay(const WINDOW *, WINDOW *);

int overwrite(const WINDOW *, WINDOW *);

EC int pair_content(short, short *, short *);

int pechochar(WINDOW *, chtype);

int pecho_wchar(WINDOW *, const cchar_t *);

int pnoutrefresh(WINDOW *, int, int, int, int, int, int);

int prefresh(WINDOW *, int, int, int, int, int, int);

int printw(const char *, ...);
### CURSES Headers

<table>
<thead>
<tr>
<th>EC</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td><code>putp(const char *)</code>;</td>
</tr>
<tr>
<td>int</td>
<td><code>putwin(WINDOW *, FILE *)</code>;</td>
</tr>
<tr>
<td>void</td>
<td><code>qiflush(void);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>raw(void);</code></td>
</tr>
<tr>
<td>EC</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td><code>redrawwin(WINDOW *)</code>;</td>
</tr>
<tr>
<td>int</td>
<td><code>refresh(void);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>reset_prog_mode(void);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>reset_shell_mode(void);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>resetty(void);</code></td>
</tr>
<tr>
<td>EC</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td><code>ripoffline(int, int (*)(WINDOW *, int));</code></td>
</tr>
<tr>
<td>int</td>
<td><code>savetty(void);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>scanw(const char *, ...);</code></td>
</tr>
<tr>
<td>EC</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td><code>scr_dump(const char *)</code>;</td>
</tr>
<tr>
<td>int</td>
<td><code>scr_init(const char *)</code>;</td>
</tr>
<tr>
<td>int</td>
<td><code>scrl(int);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>scroll(WINDOW *)</code>;</td>
</tr>
<tr>
<td>int</td>
<td><code>scrollok(WINDOW *, bool);</code></td>
</tr>
<tr>
<td>EC</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td><code>scr_restore(const char *)</code>;</td>
</tr>
<tr>
<td>int</td>
<td><code>scr_set(const char *)</code>;</td>
</tr>
<tr>
<td>int</td>
<td><code>setcchar(char_t*, const wchar_t*, const attr_t, short, const void*);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>setscrreg(int, int);</code></td>
</tr>
<tr>
<td>SCREEN</td>
<td>*set_term(SCREEN *);</td>
</tr>
<tr>
<td>int</td>
<td><code>setupterm(char *, int, int *)</code>;</td>
</tr>
<tr>
<td>EC</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td><code>slk_attr_off(const attr_t, void *);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>slk_attroff(const chtype);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>slk_attr_on(const attr_t, void *);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>slk_attroff(const chtype);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>slk_attr_set(const attr_t, short, void *);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>slk_attrset(const chtype);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>slk_clear(void);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>slk_color(short);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>slk_init(int);</code></td>
</tr>
<tr>
<td>char</td>
<td>*slk_label(int);</td>
</tr>
<tr>
<td>int</td>
<td><code>slk_noutrefresh(void);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>slk_refresh(void);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>slk_restore(void);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>slk_set(int, const char *, int);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>slk_touch(void);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>slk_wset(int, const wchar_t *, int);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>standend(void);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>standout(void);</code></td>
</tr>
<tr>
<td>EC</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td><code>start_color(void);</code></td>
</tr>
<tr>
<td>WINDOW</td>
<td>*subpad(WINDOW *, int, int, int, int);</td>
</tr>
<tr>
<td>WINDOW</td>
<td>*subwin(WINDOW *, int, int, int, int);</td>
</tr>
<tr>
<td>EC</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td><code>syncok(WINDOW *, bool);</code></td>
</tr>
<tr>
<td>chtype</td>
<td><code>termattrs(void);</code></td>
</tr>
<tr>
<td>attr_t</td>
<td><code>term_attrts(void);</code></td>
</tr>
<tr>
<td>char</td>
<td>*termname(void);</td>
</tr>
<tr>
<td>int</td>
<td><code>tigetflag(const char *);</code></td>
</tr>
<tr>
<td>int</td>
<td><code>tigetnum(const char *);</code></td>
</tr>
<tr>
<td>char</td>
<td>*tigetstr(const char *);</td>
</tr>
</tbody>
</table>
void timeout(int);
int touchline(WINDOW *, int, int);
int touchwin(WINDOW *);
char *tiparm(const char *, ...);
char *tparm(const char *, long, long, long, long, long, long, long,
            long, long, long);
int typeahead(int);
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(chtype);
int vid_puts(attr_t, short, void *, int (*)(int));
int vidputs(chtype, int (*)(int));
int vline(chtype, int);
int vline_set(const cchar_t *, int);
int vw_printw(WINDOW *, const char *, va_list);
int vw_scanw(WINDOW *, const 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 waddnstrstr(WINDOW *, const char *, int);
int waddstr(WINDOW *, const char *);
int wadd_wch(WINDOW *, const cchar_t *);
int wadd_wchstr(WINDOW *, const cchar_t *);
int wadd_wchnstr(WINDOW *, const cchar_t *, int);
int wadd_wchnstrstr(WINDOW *, const cchar_t *, int);
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);
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 wchqat(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 *, char *, int);
int  wgetn_wstr (WINDOW *, wint_t *, int);
int  wgetstr (WINDOW *, char *);

int  wget_wch (WINDOW *, wint_t *);
int  wget_wstr (WINDOW *, wint_t *);
int  whline (WINDOW *, chtype, int);
int  whline_set (WINDOW *, const cchar_t *, 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  winsertln (WINDOW *);

int  winsnstr (WINDOW *, const char *, int);
int  winsnwstr (WINDOW *, const wchar_t *, int);
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_wchstr (WINDOW *, cchar_t *);
int  winnwstr (WINDOW *, wchar_t *);

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

int  wredrawln (WINDOW *, int, int);
int  wrefresh (WINDOW *);
int  wscanfw (WINDOW *, const char *, ...);

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

void  wsyncup (WINDOW *);
void  wysyncdown (WINDOW *);

void  wtimeout (WINDOW *, int);
int  wtouchnl (WINDOW *, int, int, int);

wchar_t *wunctrl (cchar_t *);
int  wvline (WINDOW *, chtype, int);
int  wvline_set (WINDOW *, const cchar_t *, int);
APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Chapter 1, <stdbool.h> (in the XBD specification), <stdio.h> (in the XBD specification), <term.h>, <termios.h> (in the XBD specification), <unctrl.h>, <wchar.h> (in the XBD 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.

Issue 7
The prototypes for the following functions are updated:

mvscanw(), mvwscanw(), newterm(), scanw(), tigetflag(), tigetnum(), tigetstr(), tparm(), vw_printw(), vw_scanw(), wscanw()

The tparm() function has been marked obsolescent.

The tiparm() function has been added.

Corrigendum U018/3 is applied, adding the value of _XOPEN_SOURCE for environments that support the Base Specifications, Issue 5.

Corrigendum U018/5 is applied, correcting the vw_printw() function prototype.

Corrigendum U022/1 is applied, correcting the shading on the addchnstr() and addchstr() function prototypes.

Corrigendum U056/2 is applied, adding the value of _XOPEN_SOURCE for environments that support the Base Specifications, Issue 6.

Corrigendum U058/1 is applied, moving the COLOR_PAIR() and PAIR_NUMBER() functions prototypes into the “Color-Related Macros” section.
NAME
term.h — terminal capabilities

SYNOPSIS
#include <term.h>

DESCRIPTION
The following data type is defined through **typedef**:

**TERMINAL** An opaque representation of the capabilities for a single terminal from the
**terminfo** database.

The **<term.h>** header provides a declaration for the following object: **cur_term**. It represents the
current terminal record from the **terminfo** database that the application has selected by calling
**set_curterm**().

The **<term.h>** header defines the variable names listed in the **Variable** column in the table in
**Section 7.1.3** (on page 340).

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

```c
int del_curterm(TERMINAL *);
int putp(const char *);
int restartterm(char *, int, int *);
TERMINAL *set_curterm(TERMINAL *);
int setupterm(char *, int, int *);
int tigetflag(const char *);
int tigetnum(const char *);
char *tigetstr(const char *);
char *tiparm(const char *, ...);
char *tparm(const 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 as a macro:

```c
bool  As described in <stdbool.h>.
```

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Chapter 7 (on page 337), **printf()**, (in the **XSH** specification), **putp()**, **tigetflag()**, <stdbool.h> (in
the **XBD** specification)

CHANGE HISTORY
First released in Issue 4.

Issue 4, Version 2
This entry is corrected.
NAME
  unctrl.h — definitions for unctrl()

SYNOPSIS
  #include <unctrl.h>

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);

APPLICATION USAGE
  None.

RATIONALE
  None.

FUTURE DIRECTIONS
  None.

SEE ALSO
  unctrl(), <curses.h>

CHANGE HISTORY
  First released in Issue 4.
This chapter describes the Curses utilities to support applications portability and consistency of user experience at the shell command level.

The Curses utilities shall conform to the requirements stated in the XCU specification, Section 1.4, Utility Description Defaults, as if the text in the XCU specification, Section 1.4, Utility Description Defaults contained the phrase “POSIX.1-2008 or XCurses, Issue 7” instead of “POSIX.1-2008”, and contained the phrase “the Curses utilities” instead of “the standard utilities”.

The Curses utilities shall conform completely to the utility syntax guidelines defined in the XBD specification, Section 12.2, Utility Syntax Guidelines, as if those guidelines contained the term “shall” instead of “should”. On some implementations, the utilities accept usage in violation of those guidelines for backwards-compatibility as well as accepting the required form.

If a Curses utility uses operands to represent files, it is implementation-defined whether the operand ‘-’ stands for standard input (or standard output) or for a file named -.
NAME
infocmp — compare or print out terminfo descriptions

SYNOPSIS
```
EC
infocmp [--I|--L] [-1] [-A directory] [-B directory]
       [−s sortorder] [−w width] [termname]
infocmp −u [--I|--L] [-1] [-A directory] [-B directory]
       [−s sortorder] [−w width] termname termname...
infocmp [-c|−d|−n] [-A directory] [-B directory]
       [−s sortorder] [−w width] termname termname
infocmp −n [-A directory] [-B directory]
       [−s sortorder] [−w width]
```

DESCRIPTION
The infocmp utility compares a compiled terminfo entry with other terminfo entries, rewrites a terminfo description to take advantage of the use= terminfo field, or prints out a terminfo description from the compiled entry in a variety of formats.

It displays boolean fields first, then numeric fields, followed by the string fields.

If none of the −I, −L, or −n options are specified and zero or one termname is specified, the −I option is assumed. If none of the −c, −d, −n, or −u options are specified and two termname operands are specified, the −d option is assumed. If the −u option is not specified and more than two termname operands are specified, it is unspecified whether the −u option is assumed.

OPTIONS
The −d, −c, and −n options can be used for comparisons. The infocmp utility compares the terminfo description of the first terminal termname with each of the descriptions given by the entries for the other terminal’s termname. If a capability is defined for only one of the terminals, the value returned will depend on the type of the capability: F for boolean variables, −1 for integer variables, and a null string for string variables.

−d Produce a list of each capability that is different between two entries. This option is useful to show the difference between two entries, created by different people, for the same or similar terminals.

−c Produce a list of each capability that is common between two entries. Capabilities that are not set are ignored. This option can be used as a quick check to see if the −u option is worth using.

−n Produce a list of each capability that is in neither entry. If no termname is given, the environment variable TERM will be used for both of the termnames. This can be used as a quick check to see if anything was left out of a description.

The −I and −L options will produce a source listing for the terminal named by the termname operand, or for the terminal named by the environment variable TERM if no termname operand is specified.

−I Use the terminfo names.

−L Use the long C variable name listed in <term.h>.

−u Produce a terminfo source description of the first terminal termname which is relative to the sum of the descriptions given by the entries for the other terminals’ termnames. It does this by analyzing the differences between the first termname and the other termnames and producing a description with use= fields for the other terminals. In this manner, it is possible to retrofit generic terminfo entries into a
terminal’s description. Or, if two similar terminals exist, but were coded at
different times, or by different people so that each description is a full description,
using infocmp will show what can be done to change one description to be relative
to the other.

A capability is displayed with an at-sign (‘@’) if it no longer exists in the first termmame, but one
of the other termmame entries contains a value for it. A capability’s value is displayed if the value
in the first termmame is not found in any of the other termmame entries, or if the first of the other
termmame entries that has this capability gives a different value for that capability.

The order of the other termmame entries is significant. Since the terminfo compiler tic does a left-
to-right scan of the capabilities, specifying two use= entries that contain differing entries for the
same capabilities will produce different results, depending on the order in which the entries are
given. The infocmp utility will flag any such inconsistencies between the other termmame entries
as they are found.

Alternatively, specifying a capability after a use= entry that contains that capability will cause
the second specification to be ignored. Using infocmp to recreate a description can be a useful
check to make sure that everything was specified correctly in the original source description.

Another error that does not cause incorrect compiled files, but will slow down the compilation
time, is specifying superfluous use= fields. The infocmp utility will flag any superfluous use= fields.

−s sortorder  Sort the fields within each type according to the sortorder option-argument below:
  d   Leave fields in the order that they are stored in the terminfo database.
  i   Sort by terminfo name.
  l   Sort by the long C variable name.

If the −s option is not given, the fields are sorted alphabetically by the terminfo
name within each type, except in the case of the −L option, which causes the
sorting to be done by the long C variable name.

−1   Print the fields one to a line. Otherwise, the fields are printed several to a line to a
maximum width of 60 characters.

−w width  Change the output to width characters.

The location of the compiled terminfo database is taken from the environment variable
TERMINFO. If the variable is not defined, or the terminal is not found in that location, the
system terminfo database is used. The options −A and −B can be used to override this location.

−A directory  Set TERMINFO for the first termmame.

−B directory  Set TERMINFO for the other termmames. With this, it is possible to compare
descriptions for a terminal with the same name located in two different databases.
This is useful for comparing descriptions for the same terminal created by different
people.

OPERANDS
See the DESCRIPTION.

STDIN
Not used.
INPUT FILES
None.

ENVIRONMENT VARIABLES
The following environment variables shall affect the execution of infocmp:

LANG       Provide a default value for the internationalization variables that are unset or null. (See the XBD specification, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL     If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE   Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH    Determine the location of message catalogs for the processing of LC_MESSAGES.

TERM       Determine the default terminal name. If this variable is unset or null, and no termname operand is specified, the behavior is unspecified.

TERMINFO   Determine the location of a compiled terminfo database to be used instead of the system terminfo database.

ASYNCHRONOUS EVENTS
Default.

STDOUT
When the −I or −L option is specified (explicitly or implicitly), the output shall consist of the terminfo source for the specified terminal in the format described in Chapter 7 (on page 337), except that if the −L option is specified, the capabilities are identified by their long C variable names instead of the Capname short names defined in Section 7.1.3 (on page 340).

When the −d option is specified (explicitly or implicitly), the output shall contain differences between the two entries in an unspecified format.

When the −c option is specified, the output shall contain a list of capabilities common between the two entries in an unspecified format.

When the −n option is specified, the output shall contain a list of capabilities that are in neither entry in an unspecified format.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.
EXIT STATUS
The following exit values shall be returned:

0  Successful completion.
>0  An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
None.

EXAMPLES
None.

RATIONALE
Implementations of infocmp exhibit different behavior when used outside the constraints of the SYNOPSIS. In particular, the behavior is unspecified when:

• The −I or −L option is used with more than one termname operand, without −u.
• The −c, −d, or −n option is used with one termname operand or with more than two termname operands.
• Any two or more of the −I, −L, −c, −d, and −n options are used together.

FUTURE DIRECTIONS
None.

SEE ALSO
Chapter 7, tic, untic, <term.h>, the XBD specification: Section 8.2, Internationalization Variables

CHANGE HISTORY
First introduced in Issue 7. Derived from Solaris 7.
NAME

tic — translate terminfo files from source to compiled format

SYNOPSIS

tic [-c] file...

DESCRIPTION

The tic utility translates terminfo files from the source format into the compiled format.

If the TERMININFO environment variable is set, the results shall be placed there; otherwise, they
shall be placed in the system terminfo database.

The tic utility compiles all terminfo descriptions in the file or files specified by the file operand.
When the tic utility finds a use= field, it searches first the current file, then reads in the compiled
entry from the system terminfo database to complete the entry. If the environment variable
TERMININFO is set, that directory is searched instead of the system terminfo database.

The tic utility may impose limits on the size of compiled entries and on the length of the name
field. The limit on the size of compiled entries, if any, shall be at least 4096 bytes. The limit on
the length of the name field, if any, shall be at least 128 bytes. The tic utility shall support
terminal names of at least 14 bytes. Users creating portable terminfo description files should not
exceed these minimum limits

OPTIONS

-c

Check the file for errors only. Errors in the use= field need not be detected.

OPERANDS

See the DESCRIPTION.

STDIN

The standard input shall be used if a file operand is ‘-’ and the implementation treats the ‘-’
as meaning standard input. Otherwise, the standard input shall not be used. See the INPUT FILES section.

INPUT FILES

The input files shall be text files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of tic:

LANG

Provide a default value for the internationalization variables that are unset or null.
(See the XBD specification, Section 8.2, Internationalization Variables for the
precedence of internationalization variables used to determine the values of locale
categories.)

LC_ALL

If set to a non-empty string value, override the values of all the other
internationalization variables.

LC_CTYPE

Determine the locale for the interpretation of sequences of bytes of text data as
characters (for example, single-byte as opposed to multi-byte characters in
arguments).

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of
diagnostic messages written to standard error.

NLSPATH

Determine the location of message catalogs for the processing of LC_MESSAGES.
TERMINFO  Determine the location of a compiled terminfo database to be used instead of the system terminfo database.

ASYNCHRONOUS EVENTS
Default.

STDOUT
Not used.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
Compiled terminfo database entries in unspecified format are created.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0  Successful completion.

>0  An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
None.

EXAMPLES
None.

RATIONALE
Some implementations of the tic utility report an error if no file operands are specified; other implementations read terminfo descriptions from standard input or from a default file such as ./terminfo.src in this case. This standard allows the latter two behaviors as extensions, but conforming applications are required to supply one or more file operands.

FUTURE DIRECTIONS
None.

SEE ALSO
Chapter 7, infocmp, untic, the XBD specification: Section 8.2, Internationalization Variables

CHANGE HISTORY
First introduced in Issue 7. Derived from Tru64 UNIX.
NAME
tput — initialize a terminal or query terminfo database

SYNOPSIS

```
tput [-T type] capname [parm...]
tput -S
```

DESCRIPTION
When XCURSES is supported, this description for the tput utility replaces that in the XCU specification.

The `tput` utility uses the terminfo database to make the values of terminal-dependent capabilities and information available to the shell (see `sh` in the XCU specification); to clear, initialize, or reset the terminal; or to return the long name of the requested terminal type. The `tput` utility outputs a string if the capability attribute (`capname`) is of type `string`, or an integer if the attribute is of type `integer`. If the attribute is of type `boolean`, `tput` simply sets the exit status (0 for TRUE if the terminal has the capability, 1 for FALSE if it does not), and produces no output.

OPTIONS
The following options are supported:

- `-T type` Indicate the type of terminal. Normally this option is unnecessary, because the default is taken from the environment variable `TERM`. If `-T` is specified, then the environment variables `LINES` and `COLUMNS` and the layer size will not be referenced.

- `-S` Allow more than one capability per invocation of `tput`. The capabilities must be passed to `tput` from the standard input instead of from the command line (see the EXAMPLES section). Only one `capname` is allowed per line. The `-S` option changes the meaning of the 0 and 1 boolean and string exit statuses (see the EXIT STATUS section).

OPERANDS
The following operands shall be supported:

- `capname` Indicate the capability attribute from the terminfo database. See Chapter 7 (on page 337) for a complete list of capabilities and the `capname` associated with each.

  In addition, in the POSIX locale the following strings shall be supported as `capname` operands:

  - `clear` Display the clear-screen sequence.
  - `init` If the terminfo database is present and an entry for the user’s terminal exists (see `-T type` above), the following shall occur:
    1. If present, the terminal’s initialization strings shall be output (`is1`, `is2`, `is3`, `if`, `iprog`).
    2. Any delays (for instance, `<newline>`) specified in the entry shall be set in the terminal attributes (see the XBD specification, Chapter 11, General Terminal Interface).
    3. Tabs expansion shall be turned on or off according to the specification in the entry.
4. If tabs are not expanded, standard tabs shall be set (every 8 spaces).

If an entry does not contain the information needed for any of the four above activities, that activity shall be silently skipped.

reset

Instead of putting out initialization strings, the terminal’s reset strings shall be output if present \((rs1i, rs2, rs3, rf)\). If the reset strings are not present, but initialization strings are, the initialization strings shall be output. Otherwise, reset shall act identically to \texttt{init}.

longname

If the \texttt{terminfo} database is present and an entry for the user’s terminal exists (see \texttt{−T type} above), then the long name of the terminal shall be output. The long name is the last name in the name field of the terminals’ entry.

parm

If the attribute is a string that takes parameters, the argument \texttt{parm} will be instantiated into the string.

STDIN

If the \texttt{−S} option is specified, lines are read from standard input and processed as if the contents of each line had been specified as a \texttt{capname} operand followed by zero or more \texttt{parm} operands on the command line, except for the exit status.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of \texttt{tput}:

\texttt{COLUMNS}

Override the system-selected horizontal screen size. See the \texttt{XBD} specification, Chapter 8, Environment Variables for valid values and results when it is unset or null.

\texttt{LANG}

Provide a default value for the internationalization variables that are unset or null. (See the \texttt{XBD} specification, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

\texttt{LC_ALL}

If set to a non-empty string value, override the values of all the other internationalization variables.

\texttt{LC_CTYPE}

Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

\texttt{LC_MESSAGES}

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

\texttt{LINES}

Override the system-selected vertical screen size. See the \texttt{XBD} specification, Chapter 8, Environment Variables for valid values and results when it is unset or null.

\texttt{NLSPATH}

Determine the location of message catalogs for the processing of \texttt{LC_MESSAGES}.

\texttt{TERM}

Determine the terminal type. If this variable is unset or null, and if the \texttt{−T} option is not specified, an unspecified default terminal type shall be used.
ASYNCHRONOUS EVENTS
    Default.

STDOUT
    See the DESCRIPTION.

STDERR
    The standard error shall be used only for diagnostic messages.

OUTPUT FILES
    None.

EXTENDED DESCRIPTION
    None.

EXIT STATUS
    The following exit values are returned:
    0 — If capname is of type boolean and –S is not specified, indicates TRUE.
        — If capname is of type string and –S is not specified, indicates capname is defined for this terminal type.
        — If capname is of type boolean or string and –S is specified, indicates that all lines were successful.
        — capname is of type integer.
        — The requested string was written successfully.
    1 — If capname is of type boolean and –S is not specified, indicates FALSE.
        — If capname is of type string and –S is not specified, indicates that capname is not defined for this terminal type.
    2 Usage error.
    3 No information is available about the specified terminal type.
    4 The specified operand is invalid.
    255 capname is a numeric variable that is not specified in the terminfo database.

Any other value
    An error occurred.

CONSEQUENCES OF ERRORS
    Default.

APPLICATION USAGE
    None.

EXAMPLES

    Using the tput command

    This example initializes the terminal according to the type of terminal in the environment variable TERM:
    tput init

    The next example resets an AT&T 5620 terminal, overriding the type of terminal in the environment variable TERM:
tput -T 5620 reset

The following example outputs the sequence to move the cursor to row 0, column 0 (the upper left corner of the screen, usually known as the “home” cursor position):

tput cup 0 0

The next example sends the sequence to move the cursor to row 23, column 4:

tput cup 23 4

The next example outputs the clear-screen sequence for the current terminal:

tput clear

The next command outputs the number of columns for the current terminal:

tput cols

The following command outputs the number of columns for the 450 terminal:

tput -T 450 cols

The next example sets the shell variable `bold` to the begin standout mode sequence, and `offbold` to the end standout mode sequence, for the current terminal and then uses them in a prompt:

```
bold=$(tput smso)
if [ $? -ne 0 ]
then
 ...
fi
offbold=$(tput rmso)
if [ $? -ne 0 ]
then
 ...
fi
printf %s "${bold}Please type in your name: ${offbold}"
```

This example sets the exit status to indicate whether the current terminal is a hardcopy terminal:

tput hc

The next example prints the long name from the `terminfo` database for the type of terminal specified in the environment variable `TERM`:

tput longname

This last example shows `tput` processing several capabilities in one invocation. This example clears the screen, moves the cursor to position 10,10, and turns on bold (extra bright) mode. The list is terminated by an exclamation mark (`!' on a line by itself:

```
tput -S <<!
clear
cup 10 10
bold
!
```
RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Chapter 7, the XBD specification: Section 8.2, Internationalization Variables; the XCU specification: sh, stty, tabs

CHANGE HISTORY
First introduced in Issue 7. Derived from Solaris 7.
NAME
untic — terminfo de-compiler

SYNOPSIS
untic [-f file]
untic term

DESCRIPTION
The untic utility translates a terminfo file from the compiled format into the source format suitable for use by the tic utility. If the environment variable TERMINFO is set to a pathname, untic checks for a compiled terminfo description of the terminal under the path specified by TERMINFO before checking the system terminfo database. Otherwise, only the system terminfo database is checked.

Normally untic uses the terminal type obtained from the TERM environment variable. Using the term operand, however, the user can specify the terminal type used.

When the -f option is specified, the file option argument specifies the file used for translation.

The untic utility writes the de-compiled terminfo description result to standard output.

OPTIONS
- f file Specify the file to be used. This option bypasses the use of the TERM and TERMINFO environment variables.

OPERANDS
The following operand shall be supported:

term Indicate the type of terminal. If this operand is not present, the terminal is derived from the environment variable TERM.

STDIN
Not used.

INPUT FILES
The input file is a compiled terminfo database entry, either present in the system terminfo database or created by the tic utility.

ENVIRONMENT VARIABLES
The following environment variables shall affect the execution of untic:

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of LC_MESSAGES.

TERM Determine the default terminal name. If this variable is unset or null, and no term operand is specified, behavior is unspecified.

TERMINFO Determine the location of a compiled terminfo database to be used instead of the system terminfo database.
ASYNCHRONOUS EVENTS
  Default.

STDOUT
  See the DESCRIPTION.

STDERR
  The standard error shall be used only for diagnostic messages.

OUTPUT FILES
  None.

EXTENDED DESCRIPTION
  None.

EXIT STATUS
  The following exit values shall be returned:
    0  Successful completion.
    >0  An error occurred.

CONSEQUENCES OF ERRORS
  Default.

APPLICATION USAGE
  None.

EXAMPLES
  None.

RATIONALE
  None.

FUTURE DIRECTIONS
  None.

SEE ALSO
  Chapter 7, infocmp, tic, the XBD specification: Section 8.2, Internationalization Variables

CHANGE HISTORY
  First introduced in Issue 7. Derived from HPUX.
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.

The tic utility, described in Chapter 6 (on page 323), accepts source files in the format specified in this chapter and can be used to enter information into the terminfo database. 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 7.1 describes the syntax of terminfo source files. The grammar and lexical conventions appear in Section 7.1.2 (on page 338). A list of all terminal capabilities defined by The Open Group appears in Section 7.1.3 (on page 340). An example follows in Section 7.1.4 (on page 349). Section A.1 (on page 353) describes the specification of devices in general, such as video terminals. Section A.2 (on page 366) 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.

### 7.1 Source File Syntax

Source files can use the ISO 8859-1:1987 codeset. The behavior when the source file is in another codeset is unspecified. Traditional practice has been to translate information from other codesets into the source file syntax.

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 one) 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:

```
alias₁ | alias₂ | ... | aliasₙ | longname,
<white space> am, lines #24,
<white space> home=\Eeh,
```

The first line, commonly referred to as the header line, must begin in column one and must contain at least two aliases separated by vertical bars. The last field in the header line must be the long name of the device and it may contain any string.
Alias names must be unique in the terminfo database and they must conform to filenaming conventions established by implementation-defined terminfo compilation utilities. Implementations will recognize alias names consisting only of characters from the portable filename character set except that implementations need not accept a first character of minus (‘-’). For example, a typical restriction is that they cannot contain white space or slashes. There may be further constraints imposed on source file values by the implementation-defined terminfo compilation utilities. Section A.4.1 (on page 377) 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.

Whenever possible, capability names are chosen to be the same as or similar to those specified by ISO/IEC 6429: 1992. Semantics are also intended to match those of that 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>.

### 7.1.1 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.

### 7.1.2 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_LINE^ rest_of_header_line feature_lines
              | descriptions START_OF_HEADER_LINE rest_of_header_line
              | feature_lines
              ;
rest_of_header_line : PIPE LONGNAME COMMA NEWLINE
```

1. An ALIAS that begins in column one. This is handled by the lexical analyzer.
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 characters other than ‘,’ ‘/’, and ‘|’.
3. A LONGNAME may contain any print 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 ‘#’

---

2. A BOOLEAN feature that begins after column one but is the first feature on the feature line. This is handled by the lexical analyzer.
3. A NUMERIC feature that begins after column one but is the first feature on the feature line. This is handled by the lexical analyzer.
4. A STRING feature that begins after column one but is the first feature on the feature line. This is handled by the lexical analyzer.
5. Graph characters are those characters for which isgraph() returns non-zero.
6. Print characters are those characters for which isprint() returns non-zero.
b. The ‘=’ character
   c. A string which may contain any print characters other than ‘,’

7. White space immediately following a ‘,’ ‘’ is ignored.

8. Comments are lines consisting of zero or more whitespace characters followed by a ‘#’ sign, followed by zero or more non-<newline> characters and terminated by a <newline>.

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

7.1.3 Defined Capabilities

The Open Group 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 tigetflag() on page 232.

The table of capabilities has the following columns:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Capname</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_left_margin</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>Terminal has automatic margins</td>
</tr>
<tr>
<td>back_color_erase</td>
<td>bce</td>
<td>Screen erased with background color</td>
</tr>
<tr>
<td>can_change</td>
<td>ccc</td>
<td>Terminal can redefine existing color</td>
</tr>
<tr>
<td>cel_standout_glitch</td>
<td>xhp</td>
<td>Standout not erased by overwriting (hp)</td>
</tr>
<tr>
<td>col_addr_glitch</td>
<td>xhpa</td>
<td>Only positive motion for hpa/mhpa caps</td>
</tr>
<tr>
<td>cpi_changes_res</td>
<td>cpx</td>
<td>Changing character pitch changes resolution</td>
</tr>
<tr>
<td>cpi_changes_res</td>
<td>crx</td>
<td>Using cr turns off micro mode</td>
</tr>
<tr>
<td>dest_tabs_magic_smso</td>
<td>xt</td>
<td>Destructive tabs, magic smso char (t1061)</td>
</tr>
<tr>
<td>eat_newline_glitch</td>
<td>xenl</td>
<td>Newline ignored after 80 columns (Concept)</td>
</tr>
<tr>
<td>erase_overstrike</td>
<td>eo</td>
<td>Can erase overstrikes with a &lt;blank&gt;</td>
</tr>
<tr>
<td>generic_type</td>
<td>gn</td>
<td>Generic line type (e.g., dialup, switch)</td>
</tr>
<tr>
<td>hard_copy</td>
<td>hc</td>
<td>Hardcopy terminal</td>
</tr>
<tr>
<td>hard_cursor</td>
<td>chts</td>
<td>Cursor is hard to see</td>
</tr>
<tr>
<td>has_meta_key</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>Printer needs operator to change character set</td>
</tr>
<tr>
<td>has_status_line</td>
<td>hs</td>
<td>Has extra “status line”</td>
</tr>
<tr>
<td>hue_lightness_saturation</td>
<td>hls</td>
<td>Terminal uses only HLS color notation (Tektronix)</td>
</tr>
<tr>
<td>insert_null_glitch</td>
<td>in</td>
<td>Insert mode distinguishes nulls</td>
</tr>
</tbody>
</table>
### Terminfo Source Format (ENHANCED CURSES) Source File Syntax

<table>
<thead>
<tr>
<th>Variable</th>
<th>Capname</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpi_changes_res</td>
<td>lpix</td>
<td>Changing line pitch changes resolution</td>
</tr>
<tr>
<td>memory_above</td>
<td>da</td>
<td>Display may be retained above the screen</td>
</tr>
<tr>
<td>memory_below</td>
<td>db</td>
<td>Display may be retained below the screen</td>
</tr>
<tr>
<td>move_insert_mode</td>
<td>mir</td>
<td>Safe to move while in insert mode</td>
</tr>
<tr>
<td>move_standout_mode</td>
<td>msgr</td>
<td>Safe to move in standout modes</td>
</tr>
<tr>
<td>needs_xon_xoff</td>
<td>nxon</td>
<td>Padding won’t work, xon/xoff required</td>
</tr>
<tr>
<td>no_esc_ctlc</td>
<td>xsb</td>
<td>Beehive (f1=escape, f2=ctrl C)</td>
</tr>
<tr>
<td>no_pad_char</td>
<td>npc</td>
<td>Pad character doesn’t exist</td>
</tr>
<tr>
<td>non_dest_scroll_region</td>
<td>ndscr</td>
<td>Scrolling region is non-destructive</td>
</tr>
<tr>
<td>non_reo_rmcup</td>
<td>smcup</td>
<td>Does not reverse rmcup</td>
</tr>
<tr>
<td>over_strike</td>
<td>os</td>
<td>Terminal overstrikes on hard-copy terminal</td>
</tr>
<tr>
<td>prtr_silent</td>
<td>mc5i</td>
<td>Printer won’t echo on screen</td>
</tr>
<tr>
<td>row_addr_glitch</td>
<td>xvpa</td>
<td>Only positive motion for vpa/mvpa caps</td>
</tr>
<tr>
<td>semi_auto_right_margin</td>
<td>sam</td>
<td>Printing in last column causes cr</td>
</tr>
<tr>
<td>status_line_esc_ok</td>
<td>eslok</td>
<td>Escape can be used on the status line</td>
</tr>
<tr>
<td>tilda_glitch</td>
<td>hz</td>
<td>Hazeltine; can’t print tilde (˜)</td>
</tr>
<tr>
<td>transparent_underline</td>
<td>ul</td>
<td>Underline character overstrikes</td>
</tr>
<tr>
<td>xon_xoff</td>
<td>xon</td>
<td>Terminal uses xon/xoff handshaking</td>
</tr>
</tbody>
</table>

### Numbers

<table>
<thead>
<tr>
<th>Variable</th>
<th>Capname</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit_image_entwining</td>
<td>bitwin</td>
<td>Number of passes for each bit-map row</td>
</tr>
<tr>
<td>bit_image_type</td>
<td>bitype</td>
<td>Type of bit image device</td>
</tr>
<tr>
<td>buffer_capacity</td>
<td>bufsz</td>
<td>Number of bytes buffered before printing</td>
</tr>
<tr>
<td>buttons</td>
<td>btns</td>
<td>Number of buttons on the mouse</td>
</tr>
<tr>
<td>columns</td>
<td>cols</td>
<td>Number of columns in a line</td>
</tr>
<tr>
<td>dot_horz_spacing</td>
<td>spinh</td>
<td>Spacing of dots horizontally in dots per inch</td>
</tr>
<tr>
<td>dot_vert_spacing</td>
<td>spinv</td>
<td>Spacing of pins vertically in pins per inch</td>
</tr>
<tr>
<td>init_tabs</td>
<td>it</td>
<td>Initial number of columns between tab positions</td>
</tr>
<tr>
<td>label_height</td>
<td>lh</td>
<td>Number of rows in each label</td>
</tr>
<tr>
<td>label_width</td>
<td>lw</td>
<td>Number of columns in each label</td>
</tr>
<tr>
<td>lines</td>
<td>lines</td>
<td>Number of lines on a screen or a page</td>
</tr>
<tr>
<td>lines_of_memory</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>Maximum combined video attributes terminal can display</td>
</tr>
<tr>
<td>magic_cookie_glitch</td>
<td>xmc</td>
<td>Number of &lt;blank&gt; characters left by smso or rmso</td>
</tr>
<tr>
<td>max_colors</td>
<td>colors</td>
<td>Maximum number of colors on the screen</td>
</tr>
<tr>
<td>max MICRO_ADDRESS</td>
<td>maddr</td>
<td>Maximum value in micro_address</td>
</tr>
<tr>
<td>max MICRO_JUMP</td>
<td>mjjump</td>
<td>Maximum value in parm micro</td>
</tr>
<tr>
<td>max_pairs</td>
<td>pairs</td>
<td>Maximum number of color-pairs on the screen</td>
</tr>
<tr>
<td>maximum WINDOWS</td>
<td>wnum</td>
<td>Maximum number of definable windows</td>
</tr>
<tr>
<td>micro_col_size</td>
<td>mcs</td>
<td>Character step size when in micro mode</td>
</tr>
<tr>
<td>micro_line_size</td>
<td>mls</td>
<td>Line step size when in micro mode</td>
</tr>
<tr>
<td>no_color_video</td>
<td>ncv</td>
<td>Video attributes that can’t be used with colors</td>
</tr>
<tr>
<td>num_labels</td>
<td>nlab</td>
<td>Number of labels on screen (start at 1)</td>
</tr>
<tr>
<td>number_of_pins</td>
<td>npins</td>
<td>Number of pins in print-head</td>
</tr>
<tr>
<td>output_res_char</td>
<td>orc</td>
<td>Horizontal resolution in units per character</td>
</tr>
<tr>
<td>output_res_line</td>
<td>orl</td>
<td>Vertical resolution in units per line</td>
</tr>
<tr>
<td>output_res_hORIZ_INCH</td>
<td>orhi</td>
<td>Horizontal resolution in units per inch</td>
</tr>
<tr>
<td>Variable</td>
<td>Capname</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>output_res_vert_inch</td>
<td>orvi</td>
<td>Vertical resolution in units per inch</td>
</tr>
<tr>
<td>padding_baud_rate</td>
<td>pb</td>
<td>Lowest baud rate where padding needed</td>
</tr>
<tr>
<td>print_rate</td>
<td>cps</td>
<td>Print rate in characters per second</td>
</tr>
<tr>
<td>virtual_terminal</td>
<td>vt</td>
<td>Virtual terminal number</td>
</tr>
<tr>
<td>wide_char_size</td>
<td>widcs</td>
<td>Character step size when in double-wide mode</td>
</tr>
<tr>
<td>width_status_line</td>
<td>wsl</td>
<td>Number of columns in status line</td>
</tr>
</tbody>
</table>

**Strings**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Capname</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acs_chars</td>
<td>acsc</td>
<td>Graphic charset pairs aAbBcC</td>
</tr>
<tr>
<td>alt_scancode_esc</td>
<td>scesa</td>
<td>Alternate escape for scancode emulation (default is for VT100)</td>
</tr>
<tr>
<td>back_tab</td>
<td>cbt</td>
<td>Back tab</td>
</tr>
<tr>
<td>bell</td>
<td>bel</td>
<td>Audible signal (bell)</td>
</tr>
<tr>
<td>bit_image_carriage_return</td>
<td>bicr</td>
<td>Move to beginning of same row</td>
</tr>
<tr>
<td>bit_image_newline</td>
<td>binel</td>
<td>Move to next row of the bit image</td>
</tr>
<tr>
<td>bit_image_repeat</td>
<td>birep</td>
<td>Repeat bit-image cell #1 #2 times</td>
</tr>
<tr>
<td>carriage_return</td>
<td>cr</td>
<td>Carriage-return</td>
</tr>
<tr>
<td>change_char_pitch</td>
<td>cpi</td>
<td>Change number of characters per inch</td>
</tr>
<tr>
<td>change_line_pitch</td>
<td>lpi</td>
<td>Change number of lines per inch</td>
</tr>
<tr>
<td>change_res_horz</td>
<td>chr</td>
<td>Change horizontal resolution</td>
</tr>
<tr>
<td>change_res_vert</td>
<td>cvr</td>
<td>Change vertical resolution</td>
</tr>
<tr>
<td>change_scroll_region</td>
<td>csr</td>
<td>Change to lines #1 through #2 (VT100)</td>
</tr>
<tr>
<td>char_padding</td>
<td>rmp</td>
<td>Like ip but when in replace mode</td>
</tr>
<tr>
<td>char_set_names</td>
<td>csnm</td>
<td>Returns a list of character set names</td>
</tr>
<tr>
<td>clear_all_tabs</td>
<td>tbc</td>
<td>Clear all tab stops</td>
</tr>
<tr>
<td>clear_margins</td>
<td>mgc</td>
<td>Clear all margins (top, bottom, and sides)</td>
</tr>
<tr>
<td>clear_screen</td>
<td>clear</td>
<td>Clear screen and home cursor</td>
</tr>
<tr>
<td>clr_bol</td>
<td>el1</td>
<td>Clear to beginning of line, inclusive</td>
</tr>
<tr>
<td>clr_eol</td>
<td>el</td>
<td>Clear to end of line</td>
</tr>
<tr>
<td>clr_eos</td>
<td>ed</td>
<td>Clear to end of display</td>
</tr>
<tr>
<td>code_set_init</td>
<td>csin</td>
<td>Init sequence for multiple codesets</td>
</tr>
<tr>
<td>color_names</td>
<td>colornm</td>
<td>Give name for color #1</td>
</tr>
<tr>
<td>column_address</td>
<td>hpa</td>
<td>Set horizontal position to absolute #1</td>
</tr>
<tr>
<td>command_character</td>
<td>cmdch</td>
<td>Terminal settable cmd character in prototype</td>
</tr>
<tr>
<td>create_window</td>
<td>cwin</td>
<td>Define win #1 to go from #2,#3 to #4,#5</td>
</tr>
<tr>
<td>cursor_address</td>
<td>cup</td>
<td>Move to row #1 col #2</td>
</tr>
<tr>
<td>cursor_down</td>
<td>cud1</td>
<td>Down one line</td>
</tr>
<tr>
<td>cursor_home</td>
<td>home</td>
<td>Home cursor (if no cup)</td>
</tr>
<tr>
<td>cursor_invisible</td>
<td>civis</td>
<td>Make cursor invisible</td>
</tr>
<tr>
<td>cursor_left</td>
<td>cub1</td>
<td>Move left one space.</td>
</tr>
<tr>
<td>cursor_mem_address</td>
<td>mrcup</td>
<td>Memory-relative cursor addressing</td>
</tr>
<tr>
<td>cursor_normal</td>
<td>cnorm</td>
<td>Make cursor appear normal (undo cvvis/cvis)</td>
</tr>
<tr>
<td>cursor_right</td>
<td>cuf1</td>
<td>Non-destructive space (cursor or carriage right)</td>
</tr>
<tr>
<td>cursor_to_ll</td>
<td>ll</td>
<td>Last line, first column (if no cup)</td>
</tr>
<tr>
<td>cursor_up</td>
<td>cuu1</td>
<td>Upline (cursor up)</td>
</tr>
<tr>
<td>cursor_visible</td>
<td>cvvis</td>
<td>Make cursor very visible</td>
</tr>
<tr>
<td>define_bit_image_region</td>
<td>defbi</td>
<td>Define rectangular bit-image region</td>
</tr>
<tr>
<td>define_char</td>
<td>defc</td>
<td>Define a character in a character set</td>
</tr>
<tr>
<td>Variable</td>
<td>Capname</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>delete_character</td>
<td>dch1</td>
<td>Delete character</td>
</tr>
<tr>
<td>delete_line</td>
<td>dl1</td>
<td>Delete line</td>
</tr>
<tr>
<td>device_type</td>
<td>devt</td>
<td>Indicate language/codeset support</td>
</tr>
<tr>
<td>dial_phone</td>
<td>dial</td>
<td>Dial phone number #1</td>
</tr>
<tr>
<td>dis_status_line</td>
<td>dsl</td>
<td>Disable status line</td>
</tr>
<tr>
<td>display_clock</td>
<td>dclk</td>
<td>Display time-of-day clock</td>
</tr>
<tr>
<td>display_pc_char</td>
<td>dispc</td>
<td>Display PC character</td>
</tr>
<tr>
<td>down_half_line</td>
<td>hd</td>
<td>Half-line down (forward 1/2 linefeed)</td>
</tr>
<tr>
<td>ena_acs</td>
<td>enacs</td>
<td>Enable alternate character set</td>
</tr>
<tr>
<td>end_bit_image_region</td>
<td>endbi</td>
<td>End a bit-image region</td>
</tr>
<tr>
<td>enter_alt_charset_mode</td>
<td>smacs</td>
<td>Start alternate character set</td>
</tr>
<tr>
<td>enter_am_mode</td>
<td>smam</td>
<td>Turn on automatic margins</td>
</tr>
<tr>
<td>enter_blink_mode</td>
<td>blink</td>
<td>Turn on blinking</td>
</tr>
<tr>
<td>enter_bold_mode</td>
<td>bold</td>
<td>Turn on bold (extra bright) mode</td>
</tr>
<tr>
<td>enter_ca_mode</td>
<td>smcup</td>
<td>String to begin programs that use cup</td>
</tr>
<tr>
<td>enter_delete_mode</td>
<td>smdc</td>
<td>Delete mode (enter)</td>
</tr>
<tr>
<td>enter_dim_mode</td>
<td>dim</td>
<td>Turn on half-bright mode</td>
</tr>
<tr>
<td>enter_doublewide_mode</td>
<td>swidm</td>
<td>Enable double wide printing</td>
</tr>
<tr>
<td>enter_draft_quality</td>
<td>sdrfq</td>
<td>Set draft quality print</td>
</tr>
<tr>
<td>enter_horizontal_hl_mode</td>
<td>ehhlm</td>
<td>Turn on horizontal highlight mode</td>
</tr>
<tr>
<td>enter_insert_mode</td>
<td>smir</td>
<td>Insert mode (enter)</td>
</tr>
<tr>
<td>enter_italics_mode</td>
<td>sitm</td>
<td>Enable italics</td>
</tr>
<tr>
<td>enter_left_hl_mode</td>
<td>elhlm</td>
<td>Turn on left highlight mode</td>
</tr>
<tr>
<td>enter_leftward_mode</td>
<td>slm</td>
<td>Enable leftward carriage motion</td>
</tr>
<tr>
<td>enter_low_hl_mode</td>
<td>elohlm</td>
<td>Turn on low highlight mode</td>
</tr>
<tr>
<td>enter_micro_mode</td>
<td>smicm</td>
<td>Enable micro motion capabilities</td>
</tr>
<tr>
<td>enter_near_letter_quality</td>
<td>snlq</td>
<td>Set near-letter quality print</td>
</tr>
<tr>
<td>enter_normal_quality</td>
<td>snrmq</td>
<td>Set normal quality print</td>
</tr>
<tr>
<td>enter_pc_charset_mode</td>
<td>smpch</td>
<td>Enter PC character display mode</td>
</tr>
<tr>
<td>enter_protected_mode</td>
<td>prot</td>
<td>Turn on protected mode</td>
</tr>
<tr>
<td>enter_reverse_mode</td>
<td>rev</td>
<td>Turn on reverse video mode</td>
</tr>
<tr>
<td>enter_right_hl_mode</td>
<td>erhlm</td>
<td>Turn on right highlight mode</td>
</tr>
<tr>
<td>enter_scancode_mode</td>
<td>smsc</td>
<td>Enter PC scancode mode</td>
</tr>
<tr>
<td>enter_secure_mode</td>
<td>invis</td>
<td>Turn on blank mode (characters invisible)</td>
</tr>
<tr>
<td>enter_shadow_mode</td>
<td>sshm</td>
<td>Enable shadow printing</td>
</tr>
<tr>
<td>enter_standout_mode</td>
<td>smso</td>
<td>Begin standout mode</td>
</tr>
<tr>
<td>enter_subscript_mode</td>
<td>ssbsubm</td>
<td>Enable subscript printing</td>
</tr>
<tr>
<td>enter_superscript_mode</td>
<td>ssupm</td>
<td>Enable superscript printing</td>
</tr>
<tr>
<td>enter_top_hl_mode</td>
<td>ethlm</td>
<td>Turn on top highlight mode</td>
</tr>
<tr>
<td>enter_underline_mode</td>
<td>smul</td>
<td>Start underscore mode</td>
</tr>
<tr>
<td>enter_upward_mode</td>
<td>sum</td>
<td>Enable upward carriage motion</td>
</tr>
<tr>
<td>enter_vertical_hl_mode</td>
<td>evhlm</td>
<td>Turn on vertical highlight mode</td>
</tr>
<tr>
<td>enter_xon_mode</td>
<td>smxon</td>
<td>Turn on xon/xoff handshaking</td>
</tr>
<tr>
<td>erase_chars</td>
<td>ech</td>
<td>Erase #1 characters</td>
</tr>
<tr>
<td>exit_alt_charset_mode</td>
<td>rmacs</td>
<td>End alternate character set</td>
</tr>
<tr>
<td>exit_am_mode</td>
<td>rmam</td>
<td>Turn off automatic margins</td>
</tr>
<tr>
<td>exit_attribute_mode</td>
<td>sgr0</td>
<td>Turn off all attributes</td>
</tr>
<tr>
<td>exit_ca_mode</td>
<td>rmcup</td>
<td>String to end programs that use cup</td>
</tr>
<tr>
<td>exit_delete_mode</td>
<td>rmdc</td>
<td>End delete mode</td>
</tr>
<tr>
<td>exit_doublewide_mode</td>
<td>rwidm</td>
<td>Disable double wide printing</td>
</tr>
<tr>
<td>exit_insert_mode</td>
<td>rmir</td>
<td>End insert mode</td>
</tr>
<tr>
<td>Variable</td>
<td>Capname</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>exit_italics_mode</td>
<td>rtm</td>
<td>Disable italics</td>
</tr>
<tr>
<td>exit_leftward_mode</td>
<td>rlm</td>
<td>Enable rightward (normal) carriage motion</td>
</tr>
<tr>
<td>exit_micro_mode</td>
<td>rmicm</td>
<td>Disable micro motion capabilities</td>
</tr>
<tr>
<td>exit_pc_charset_mode</td>
<td>rmpch</td>
<td>Disable PC character display mode</td>
</tr>
<tr>
<td>exit_scancode_mode</td>
<td>rmsc</td>
<td>Disable PC scancode mode</td>
</tr>
<tr>
<td>exit_shadow_mode</td>
<td>rshtm</td>
<td>Disable shadow printing</td>
</tr>
<tr>
<td>exit_standout_mode</td>
<td>rmso</td>
<td>End standout mode</td>
</tr>
<tr>
<td>exit_subscript_mode</td>
<td>rsubm</td>
<td>Disable subscript printing</td>
</tr>
<tr>
<td>exit_superscript_mode</td>
<td>rsupm</td>
<td>Disable superscript printing</td>
</tr>
<tr>
<td>exit_underline_mode</td>
<td>rmul</td>
<td>End underscore mode</td>
</tr>
<tr>
<td>exit_upward_mode</td>
<td>rum</td>
<td>Enable downward (normal) carriage motion</td>
</tr>
<tr>
<td>exit_xon_mode</td>
<td>rmxon</td>
<td>Turn off xon/xoff handshaking</td>
</tr>
<tr>
<td>fixed_pause</td>
<td>pause</td>
<td>Pause for 2-3 seconds</td>
</tr>
<tr>
<td>flash_hook</td>
<td>hook</td>
<td>Flash the switch hook</td>
</tr>
<tr>
<td>flash_screen</td>
<td>flash</td>
<td>Visible bell (may move cursor)</td>
</tr>
<tr>
<td>form_feed</td>
<td>ff</td>
<td>Hardcopy terminal page eject</td>
</tr>
<tr>
<td>from_status_line</td>
<td>fsl</td>
<td>Return from status line</td>
</tr>
<tr>
<td>get_mouse</td>
<td>getm</td>
<td>Curses should get button events</td>
</tr>
<tr>
<td>goto_window</td>
<td>wingo</td>
<td>Go to window #1</td>
</tr>
<tr>
<td>hangup</td>
<td>hup</td>
<td>Hang-up phone</td>
</tr>
<tr>
<td>init_1string</td>
<td>is1</td>
<td>Terminal or printer initialization string</td>
</tr>
<tr>
<td>init_2string</td>
<td>is2</td>
<td>Terminal or printer initialization string</td>
</tr>
<tr>
<td>init_3string</td>
<td>is3</td>
<td>Terminal or printer initialization string</td>
</tr>
<tr>
<td>init_file</td>
<td>if</td>
<td>Name of initialization file</td>
</tr>
<tr>
<td>init_prog</td>
<td>iprog</td>
<td>Path name of program for initialization</td>
</tr>
<tr>
<td>initialize_color</td>
<td>initc</td>
<td>Set color #1 to RGB #2, #3, #4</td>
</tr>
<tr>
<td>initialize_pair</td>
<td>initp</td>
<td>Set color-pair #1 to fg #2, bg #3</td>
</tr>
<tr>
<td>insert_character</td>
<td>ich1</td>
<td>Insert character</td>
</tr>
<tr>
<td>insert_line</td>
<td>il1</td>
<td>Add new blank line</td>
</tr>
<tr>
<td>insert_padding</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>Capname</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key_dc</td>
<td>kdch1</td>
<td>Sent by delete-character key</td>
</tr>
<tr>
<td>key_dl</td>
<td>kdl1</td>
<td>Sent by delete-line key</td>
</tr>
<tr>
<td>key_down</td>
<td>kcud1</td>
<td>Sent by terminal down-arrow key</td>
</tr>
<tr>
<td>key_eic</td>
<td>krmir</td>
<td>Sent by \texttt{rmir} or \texttt{smir} in insert mode</td>
</tr>
<tr>
<td>key_end</td>
<td>kend</td>
<td>Sent by end key</td>
</tr>
<tr>
<td>key_exit</td>
<td>kext</td>
<td>Sent by exit key</td>
</tr>
<tr>
<td>key_f0</td>
<td>kf0</td>
<td>Sent by function key f0</td>
</tr>
<tr>
<td>key_f1</td>
<td>kf1</td>
<td>Sent by function key f1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>similarly for f2 through f61</td>
</tr>
<tr>
<td>key_f62</td>
<td>kf62</td>
<td>Sent by function key f62</td>
</tr>
<tr>
<td>key_f63</td>
<td>kf63</td>
<td>Sent by function key f63</td>
</tr>
<tr>
<td>key_find</td>
<td>kfnd</td>
<td>Sent by find key</td>
</tr>
<tr>
<td>key_help</td>
<td>khlp</td>
<td>Sent by help key</td>
</tr>
<tr>
<td>key_home</td>
<td>khome</td>
<td>Sent by home key</td>
</tr>
<tr>
<td>key_ic</td>
<td>kich1</td>
<td>Sent by ins-char/enter ins-mode key</td>
</tr>
<tr>
<td>key_il</td>
<td>kil1</td>
<td>Sent by insert-line key</td>
</tr>
<tr>
<td>key_left</td>
<td>kcub1</td>
<td>Sent by terminal left-arrow key</td>
</tr>
<tr>
<td>key_ll</td>
<td>kll</td>
<td>Sent by home-down key</td>
</tr>
<tr>
<td>key_mark</td>
<td>kmrk</td>
<td>Sent by mark key</td>
</tr>
<tr>
<td>key_message</td>
<td>kmsg</td>
<td>Sent by message key</td>
</tr>
<tr>
<td>key_mouse</td>
<td>kmous</td>
<td>Sent by \texttt{0631}, Mouse event has occurred</td>
</tr>
<tr>
<td>key_move</td>
<td>kmov</td>
<td>Sent by move key</td>
</tr>
<tr>
<td>key_next</td>
<td>knxt</td>
<td>Sent by next-object key</td>
</tr>
<tr>
<td>key_npage</td>
<td>knp</td>
<td>Sent by next-page key</td>
</tr>
<tr>
<td>key_open</td>
<td>kopn</td>
<td>Sent by open key</td>
</tr>
<tr>
<td>key_options</td>
<td>kopt</td>
<td>Sent by options key</td>
</tr>
<tr>
<td>key_ppage</td>
<td>kpp</td>
<td>Sent by previous-page key</td>
</tr>
<tr>
<td>key_previous</td>
<td>kprv</td>
<td>Sent by previous-object key</td>
</tr>
<tr>
<td>key_print</td>
<td>kprt</td>
<td>Sent by print or copy key</td>
</tr>
<tr>
<td>key_redo</td>
<td>krdo</td>
<td>Sent by redo key</td>
</tr>
<tr>
<td>key_reference</td>
<td>kref</td>
<td>Sent by ref(erence) key</td>
</tr>
<tr>
<td>key_refresh</td>
<td>krfr</td>
<td>Sent by refresh key</td>
</tr>
<tr>
<td>key_replace</td>
<td>krpl</td>
<td>Sent by replace key</td>
</tr>
<tr>
<td>key_restart</td>
<td>krst</td>
<td>Sent by restart key</td>
</tr>
<tr>
<td>key_resume</td>
<td>kres</td>
<td>Sent by resume key</td>
</tr>
<tr>
<td>key_right</td>
<td>kcuf1</td>
<td>Sent by terminal right-arrow key</td>
</tr>
<tr>
<td>key_save</td>
<td>ksav</td>
<td>Sent by save key</td>
</tr>
<tr>
<td>key_sbeg</td>
<td>kBEG</td>
<td>Sent by shifted beginning key</td>
</tr>
<tr>
<td>key_scancel</td>
<td>kCAN</td>
<td>Sent by shifted cancel key</td>
</tr>
<tr>
<td>key_command</td>
<td>kCMD</td>
<td>Sent by shifted command key</td>
</tr>
<tr>
<td>key_scopy</td>
<td>kCPY</td>
<td>Sent by shifted copy key</td>
</tr>
<tr>
<td>key_screate</td>
<td>kCRT</td>
<td>Sent by shifted create key</td>
</tr>
<tr>
<td>key_sdc</td>
<td>kDC</td>
<td>Sent by shifted delete-char key</td>
</tr>
<tr>
<td>key_sdl</td>
<td>kDL</td>
<td>Sent by shifted delete-line key</td>
</tr>
<tr>
<td>key_select</td>
<td>kslt</td>
<td>Sent by select key</td>
</tr>
<tr>
<td>key_send</td>
<td>kEND</td>
<td>Sent by shifted end key</td>
</tr>
<tr>
<td>Variable</td>
<td>Capname</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>key_seol</td>
<td>kEOL</td>
<td>Sent by shifted clear-line key</td>
</tr>
<tr>
<td>key_sexit</td>
<td>kEXT</td>
<td>Sent by shifted exit key</td>
</tr>
<tr>
<td>key_sf</td>
<td>kind</td>
<td>Sent by scroll-forward/down key</td>
</tr>
<tr>
<td>key_sfind</td>
<td>kFND</td>
<td>Sent by shifted find key</td>
</tr>
<tr>
<td>key_shelp</td>
<td>kHLP</td>
<td>Sent by shifted help key</td>
</tr>
<tr>
<td>key_shome</td>
<td>kHOM</td>
<td>Sent by shifted home key</td>
</tr>
<tr>
<td>key_sic</td>
<td>kIC</td>
<td>Sent by shifted input key</td>
</tr>
<tr>
<td>key_sleft</td>
<td>kLFT</td>
<td>Sent by shifted left-arrow key</td>
</tr>
<tr>
<td>key_smessage</td>
<td>kMSG</td>
<td>Sent by shifted message key</td>
</tr>
<tr>
<td>key_smove</td>
<td>kMOV</td>
<td>Sent by shifted move key</td>
</tr>
<tr>
<td>key_snext</td>
<td>kNXT</td>
<td>Sent by shifted next key</td>
</tr>
<tr>
<td>key_soptions</td>
<td>kOPT</td>
<td>Sent by shifted options key</td>
</tr>
<tr>
<td>key_sprevious</td>
<td>kPRV</td>
<td>Sent by shifted prev key</td>
</tr>
<tr>
<td>key_sprint</td>
<td>kPRT</td>
<td>Sent by shifted print key</td>
</tr>
<tr>
<td>key_sr</td>
<td>kri</td>
<td>Sent by scroll-backward/up key</td>
</tr>
<tr>
<td>key_sredo</td>
<td>kRDO</td>
<td>Sent by shifted redo key</td>
</tr>
<tr>
<td>key_sreplace</td>
<td>kRPL</td>
<td>Sent by shifted replace key</td>
</tr>
<tr>
<td>key_sright</td>
<td>kRIT</td>
<td>Sent by shifted right-arrow key</td>
</tr>
<tr>
<td>key_sresume</td>
<td>kRES</td>
<td>Sent by shifted resume key</td>
</tr>
<tr>
<td>key_ssave</td>
<td>kSAV</td>
<td>Sent by shifted save key</td>
</tr>
<tr>
<td>key_ssuspend</td>
<td>kSPD</td>
<td>Sent by shifted suspend key</td>
</tr>
<tr>
<td>key_stab</td>
<td>khts</td>
<td>Sent by set-tab key</td>
</tr>
<tr>
<td>key_sundo</td>
<td>kUND</td>
<td>Sent by shifted undo key</td>
</tr>
<tr>
<td>key_suspend</td>
<td>kspd</td>
<td>Sent by suspend key</td>
</tr>
<tr>
<td>key_undo</td>
<td>kund</td>
<td>Sent by undo key</td>
</tr>
<tr>
<td>key_up</td>
<td>kcuu1</td>
<td>Sent by terminal up-arrow key</td>
</tr>
<tr>
<td>keypad_local</td>
<td>rmxx</td>
<td>Out of “keypad-transmit” mode</td>
</tr>
<tr>
<td>keypad_xmit</td>
<td>smxx</td>
<td>Put terminal in “keypad-transmit” mode</td>
</tr>
<tr>
<td>lab_f0</td>
<td>lf0</td>
<td>Labels on function key f0 if not f0</td>
</tr>
<tr>
<td>lab_f1</td>
<td>lf1</td>
<td>Labels on function key f1 if not f1</td>
</tr>
<tr>
<td>lab_f2</td>
<td>lf2</td>
<td>Labels on function key f2 if not f2</td>
</tr>
<tr>
<td>lab_f3</td>
<td>lf3</td>
<td>Labels on function key f3 if not f3</td>
</tr>
<tr>
<td>lab_f4</td>
<td>lf4</td>
<td>Labels on function key f4 if not f4</td>
</tr>
<tr>
<td>lab_f5</td>
<td>lf5</td>
<td>Labels on function key f5 if not f5</td>
</tr>
<tr>
<td>lab_f6</td>
<td>lf6</td>
<td>Labels on function key f6 if not f6</td>
</tr>
<tr>
<td>lab_f7</td>
<td>lf7</td>
<td>Labels on function key f7 if not f7</td>
</tr>
<tr>
<td>lab_f8</td>
<td>lf8</td>
<td>Labels on function key f8 if not f8</td>
</tr>
<tr>
<td>lab_f9</td>
<td>lf9</td>
<td>Labels on function key f9 if not f9</td>
</tr>
<tr>
<td>lab_f10</td>
<td>lf10</td>
<td>Labels on function key f10 if not f10</td>
</tr>
<tr>
<td>label_format</td>
<td>fln</td>
<td>Label format</td>
</tr>
<tr>
<td>label_off</td>
<td>rmln</td>
<td>Turn off soft labels</td>
</tr>
<tr>
<td>label_on</td>
<td>smln</td>
<td>Turn on soft labels</td>
</tr>
<tr>
<td>meta_off</td>
<td>smm</td>
<td>Turn off “meta mode”</td>
</tr>
<tr>
<td>meta_on</td>
<td>smm</td>
<td>Turn on “meta mode” (8th bit)</td>
</tr>
<tr>
<td>micro_column_address</td>
<td>mhpa</td>
<td>Like column_address for micro adjustment</td>
</tr>
<tr>
<td>micro_down</td>
<td>mcud1</td>
<td>Like cursor_down for micro adjustment</td>
</tr>
<tr>
<td>micro_left</td>
<td>mcub1</td>
<td>Like cursor_left for micro adjustment</td>
</tr>
<tr>
<td>micro_right</td>
<td>mcuf1</td>
<td>Like cursor_right for micro adjustment</td>
</tr>
<tr>
<td>micro_row_address</td>
<td>mvpa</td>
<td>Like row_address for micro adjustment</td>
</tr>
<tr>
<td>micro_up</td>
<td>mcuu1</td>
<td>Like cursor_up for micro adjustment</td>
</tr>
<tr>
<td>mouse_info</td>
<td>minfo</td>
<td>Mouse status information</td>
</tr>
<tr>
<td>Variable</td>
<td>Capname</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>newline</td>
<td>nel</td>
<td>Newline (behaves like cr followed by lf)</td>
</tr>
<tr>
<td>order_of_pins</td>
<td>porder</td>
<td>Matches software bits to print-head pins</td>
</tr>
<tr>
<td>orig_colors</td>
<td>oc</td>
<td>Set all color(-pair)s to the original ones</td>
</tr>
<tr>
<td>orig_pair</td>
<td>op</td>
<td>Set default color-pair to the original one</td>
</tr>
<tr>
<td>pad_char</td>
<td>pad</td>
<td>Pad character (rather than null)</td>
</tr>
<tr>
<td>parm_dch</td>
<td>dch</td>
<td>Delete #1 chars</td>
</tr>
<tr>
<td>parm_delete_line</td>
<td>dl</td>
<td>Delete #1 lines</td>
</tr>
<tr>
<td>parm_down_cursor</td>
<td>cud</td>
<td>Like parm_down_cursor for micro adjust.</td>
</tr>
<tr>
<td>parm_down_micro</td>
<td>mcud</td>
<td>Move down #1 lines</td>
</tr>
<tr>
<td>parm_index</td>
<td>indn</td>
<td>Scroll forward #1 lines</td>
</tr>
<tr>
<td>parm_insert_line</td>
<td>il</td>
<td>Add #1 new blank lines</td>
</tr>
<tr>
<td>parm_left_cursor</td>
<td>cub</td>
<td>Move cursor left #1 spaces</td>
</tr>
<tr>
<td>parm_left_micro</td>
<td>mcub</td>
<td>Like parm_left_cursor for micro adjust.</td>
</tr>
<tr>
<td>parm_right_cursor</td>
<td>cuf</td>
<td>Move right #1 spaces</td>
</tr>
<tr>
<td>parm_right микро</td>
<td>mcuf</td>
<td>Like parm_right_cursor for micro adjust.</td>
</tr>
<tr>
<td>parm_rindex</td>
<td>rin</td>
<td>Scroll backward #1 lines</td>
</tr>
<tr>
<td>parm_up_cursor</td>
<td>cuu</td>
<td>Move cursor up #1 lines</td>
</tr>
<tr>
<td>parm_up_micro</td>
<td>mcuu</td>
<td>Like parm_up_cursor for micro adjust.</td>
</tr>
<tr>
<td>pc_term_options</td>
<td>pctrm</td>
<td>PC terminal options</td>
</tr>
<tr>
<td>pkey_key</td>
<td>pfkey</td>
<td>Prog funct key #1 to type string #2</td>
</tr>
<tr>
<td>pkey_local</td>
<td>pfloc</td>
<td>Prog funct key #1 to execute string #2</td>
</tr>
<tr>
<td>pkey_plab</td>
<td>pxl</td>
<td>Prog key #1 to xmit string #2 and show string #3</td>
</tr>
<tr>
<td>pkey_xmit</td>
<td>px</td>
<td>Prog funct key #1 to xmit string #2</td>
</tr>
<tr>
<td>plab_norm</td>
<td>pln</td>
<td>Prog label #1 to show string #2</td>
</tr>
<tr>
<td>print_screen</td>
<td>mc0</td>
<td>Print contents of the screen</td>
</tr>
<tr>
<td>prtr_non</td>
<td>mc5p</td>
<td>Turn on the printer for #1 bytes</td>
</tr>
<tr>
<td>prtr_off</td>
<td>mc4</td>
<td>Turn off the printer</td>
</tr>
<tr>
<td>prtr_on</td>
<td>mc5</td>
<td>Turn on the printer</td>
</tr>
<tr>
<td>pulse</td>
<td>pulse</td>
<td>Select pulse dialing</td>
</tr>
<tr>
<td>quick_dial</td>
<td>qdi</td>
<td>Dial phone number #1, without progress detection</td>
</tr>
<tr>
<td>remove_clock</td>
<td>rmc1k</td>
<td>Remove time-of-day clock</td>
</tr>
<tr>
<td>repeat_char</td>
<td>rep</td>
<td>Repeat char #1 #2 times</td>
</tr>
<tr>
<td>req_for_input</td>
<td>rfi</td>
<td>Send next input char (for pts)</td>
</tr>
<tr>
<td>req_mouse_pos</td>
<td>reqmp</td>
<td>Request mouse position report</td>
</tr>
<tr>
<td>reset_1string</td>
<td>rs1</td>
<td>Reset terminal completely to sane modes</td>
</tr>
<tr>
<td>reset_2string</td>
<td>rs2</td>
<td>Reset terminal completely to sane modes</td>
</tr>
<tr>
<td>reset_3string</td>
<td>rs3</td>
<td>Reset terminal completely to sane modes</td>
</tr>
<tr>
<td>reset_file</td>
<td>rf</td>
<td>Name of file containing reset string</td>
</tr>
<tr>
<td>restore_cursor</td>
<td>rc</td>
<td>Restore cursor to position of last sc</td>
</tr>
<tr>
<td>row_address</td>
<td>vpa</td>
<td>Set vertical position to absolute #1</td>
</tr>
<tr>
<td>save_cursor</td>
<td>sc</td>
<td>Save cursor position</td>
</tr>
<tr>
<td>scancode_escape</td>
<td>scesc</td>
<td>Escape for scancode emulation</td>
</tr>
<tr>
<td>scroll_forward</td>
<td>ind</td>
<td>Scroll text up</td>
</tr>
<tr>
<td>scroll_reverse</td>
<td>ri</td>
<td>Scroll text down</td>
</tr>
<tr>
<td>select_char_set</td>
<td>scs</td>
<td>Select character set</td>
</tr>
<tr>
<td>set0_des_seq</td>
<td>s0ds</td>
<td>Shift into codeset 0 (EUC set 0, ASCII)</td>
</tr>
<tr>
<td>set1_des_seq</td>
<td>s1ds</td>
<td>Shift into codeset 1</td>
</tr>
<tr>
<td>set2_des_seq</td>
<td>s2ds</td>
<td>Shift into codeset 2</td>
</tr>
<tr>
<td>set3_des_seq</td>
<td>s3ds</td>
<td>Shift into codeset 3</td>
</tr>
<tr>
<td>set_a_attributes</td>
<td>sgr1</td>
<td>Define second set of video attributes #1-#6</td>
</tr>
<tr>
<td>Variable</td>
<td>Capname</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>set_a_background</td>
<td>setab</td>
<td>Set background color to #1 using ANSI escape</td>
</tr>
<tr>
<td>set_a_foreground</td>
<td>setaf</td>
<td>Set foreground color to #1 using ANSI escape</td>
</tr>
<tr>
<td>set_attributes</td>
<td>sgr</td>
<td>Define first set of video attributes #1-#9</td>
</tr>
<tr>
<td>set_background</td>
<td>setb</td>
<td>Set background color to #1</td>
</tr>
<tr>
<td>set_bottom_margin</td>
<td>smgb</td>
<td>Set bottom margin at current line</td>
</tr>
<tr>
<td>set_bottom_margin_parm</td>
<td>smgbp</td>
<td>Set bottom margin at line #1 or #2 lines from bottom</td>
</tr>
<tr>
<td>set_clock</td>
<td>sclk</td>
<td>Set clock to hours (#1), minutes (#2), seconds (#3)</td>
</tr>
<tr>
<td>set_color_band</td>
<td>setcolor</td>
<td>Change to ribbon color #1</td>
</tr>
<tr>
<td>set_color_pair</td>
<td>scp</td>
<td>Set current color pair to #1</td>
</tr>
<tr>
<td>set_foreground</td>
<td>setf</td>
<td>Set foreground color to #1</td>
</tr>
<tr>
<td>set_left_margin</td>
<td>smgl</td>
<td>Set left margin at current column</td>
</tr>
<tr>
<td>set_left_margin_parm</td>
<td>smglp</td>
<td>Set left (right) margin at column #1 (#2)</td>
</tr>
<tr>
<td>set_lr_margin</td>
<td>smglr</td>
<td>Sets both left and right margins</td>
</tr>
<tr>
<td>set_page_length</td>
<td>slines</td>
<td>Set page length to #1 lines</td>
</tr>
<tr>
<td>set_right_margin</td>
<td>smgr</td>
<td>Set right margin at current column</td>
</tr>
<tr>
<td>set_right_margin_parm</td>
<td>smgrp</td>
<td>Set right margin at column #1</td>
</tr>
<tr>
<td>set_tab</td>
<td>hts</td>
<td>Set a tab in all rows, current column</td>
</tr>
<tr>
<td>set_tb_margin</td>
<td>smgtb</td>
<td>Sets both top and bottom margins</td>
</tr>
<tr>
<td>set_top_margin</td>
<td>smgt</td>
<td>Set top margin at current line</td>
</tr>
<tr>
<td>set_top_margin_parm</td>
<td>smgtp</td>
<td>Set top (bottom) margin at line #1 (#2)</td>
</tr>
<tr>
<td>set_window</td>
<td>wind</td>
<td>Current window is lines #1-#2 cols #3-#4</td>
</tr>
<tr>
<td>start_bit_image</td>
<td>sbim</td>
<td>Start printing bit image graphics</td>
</tr>
<tr>
<td>start_char_set_def</td>
<td>scsd</td>
<td>Start definition of a character set</td>
</tr>
<tr>
<td>stop_bit_image</td>
<td>rbim</td>
<td>End printing bit image graphics</td>
</tr>
<tr>
<td>stop_char_set_def</td>
<td>rcsd</td>
<td>End definition of a character set</td>
</tr>
<tr>
<td>subscript_characters</td>
<td>subs</td>
<td>List of “subscript-able” characters</td>
</tr>
<tr>
<td>superscript_characters</td>
<td>supcs</td>
<td>List of “superscript-able” characters</td>
</tr>
<tr>
<td>tab</td>
<td>ht</td>
<td>Tab to next 8-space hardware tab stop</td>
</tr>
<tr>
<td>these_cause_cr</td>
<td>docr</td>
<td>Printing any of these chars causes cr</td>
</tr>
<tr>
<td>to_status_line</td>
<td>tsl</td>
<td>Go to status line, col #1</td>
</tr>
<tr>
<td>tone</td>
<td>tone</td>
<td>Select touch tone dialing</td>
</tr>
<tr>
<td>user0</td>
<td>u0</td>
<td>User string 0</td>
</tr>
<tr>
<td>user1</td>
<td>u1</td>
<td>User string 1</td>
</tr>
<tr>
<td>user2</td>
<td>u2</td>
<td>User string 2</td>
</tr>
<tr>
<td>user3</td>
<td>u3</td>
<td>User string 3</td>
</tr>
<tr>
<td>user4</td>
<td>u4</td>
<td>User string 4</td>
</tr>
<tr>
<td>user5</td>
<td>u5</td>
<td>User string 5</td>
</tr>
<tr>
<td>user6</td>
<td>u6</td>
<td>User string 6</td>
</tr>
<tr>
<td>user7</td>
<td>u7</td>
<td>User string 7</td>
</tr>
<tr>
<td>user8</td>
<td>u8</td>
<td>User string 8</td>
</tr>
<tr>
<td>user9</td>
<td>u9</td>
<td>User string 9</td>
</tr>
<tr>
<td>underline_character</td>
<td>uc</td>
<td>Underscore one char and move past it</td>
</tr>
<tr>
<td>up_half_line</td>
<td>hu</td>
<td>Half-line up (reverse 1/2 linefeed)</td>
</tr>
<tr>
<td>wait_tone</td>
<td>wait</td>
<td>Wait for dial tone</td>
</tr>
<tr>
<td>xoff_character</td>
<td>xoffc</td>
<td>X-off character</td>
</tr>
<tr>
<td>xon_character</td>
<td>xonc</td>
<td>X-on character</td>
</tr>
<tr>
<td>zero_motion</td>
<td>zerom</td>
<td>No motion for the subsequent character</td>
</tr>
</tbody>
</table>
Terminfo Source Format (ENHANCED CURSES)

7.1.4

Source File Syntax

Sample Entry
The following entry describes the AT&T 610 terminal:
610 | 610bct | ATT610 | att610 | AT&T 610; 80 column; 98key keyboard,
am, eslok, hs, mir, msgr, xenl, xon,
cols#80, it#8, lh#2, lines#24, lw#8, nlab#8, wsl#80,
acsc=‘‘aaffggjjkkllmmnnooppqqrrssttuuvvwwxxyyzz{{||}}˜˜,
bel=ˆG, blink=\E[5m, bold=\E[1m, cbt=\E[Z,
civis=\E[?25l, clear=\E[H\E[J, cnorm=\E[?25h\E[?12l,
cr=\r, csr=\E[%i%p1%d;%p2%dr, cub=\E[%p1%dD, cub1=\b,
cud=\E[%p1%dB, cud1=\E[B, cuf=\E[%p1%dC, cuf1=\E[C,
cup=\E[%i%p1%d;%p2%dH, cuu=\E[%p1%dA, cuu1=\E[A,
cvvis=\E[?12;25h, dch=\E[%p1%dP, dch1=\E[P, dim=\E[2m,
dl=\E[%p1%dM, dl1=\E[M, ed=\E[J, el=\E[K, el1=\E[1K,
flash=\E[?5h$<200>\E[?5l, fsl=\E8, home=\E[H, ht=\t,
ich=\E[%p1%d@, il=\E[%p1%dL, il1=\E[L, ind=\ED, .ind=\ED$<9>,
invis=\E[8m,
is1=\E[8;0 | \E[?3;4;5;13;15l\E[13;20l\E[?7h\E[12h\E(B\E)0,
is2=\E[0mˆO, is3=\E(B\E)0, kLFT=\E[\s@, kRIT=\E[\sA,
kbs=ˆH, kcbt=\E[Z, kclr=\E[2J, kcub1=\E[D, kcud1=\E[B,
kcuf1=\E[C, kcuu1=\E[A, kfP=\EOc, kfP0=\ENp,
kfP1=\ENq, kfP2=\ENr, kfP3=\ENs, kfP4=\ENt, kfI=\EOd,
kfB=\EOe, kf4=\EOf, kf(CW=\EOg, kf6=\EOh, kf7=\EOi,
kf8=\EOj, kf9=\ENo, khome=\E[H, kind=\E[S, kri=\E[T,
ll=\E[24H, mc4=\E[?4i, mc5=\E[?5i, nel=\EE,
pfxl=\E[%p1%d;%p2%l%02dq%?%p1%{9}%<%t\s\s\sF%p1%1d\s\s\s\s\s
\s\s\s\s\s\s%;%p2%s,
pln=\E[%p1%d;0;0;0q%p2%:-16.16s, rc=\E8, rev=\E[7m,
ri=\EM, rmacs=ˆO, rmir=\E[4l, rmln=\E[2p, rmso=\E[m,
rmul=\E[m, rs2=\Ec\E[?3l, sc=\E7,
sgr=\E[0%?%p6%t;1%;%?%p5%t;2%;%?%p2%t;4%;%?%p4%t;5%;
%?%p3%p1% | %t;7%;%?%p7%t;8%;m%?%p9%tˆN%eˆO%;,
sgr0=\E[mˆO, smacs=ˆN, smir=\E[4h, smln=\E[p,
smso=\E[7m, smul=\E[4m, tsl=\E7\E[25;%i%p1%dx,

7.1.5

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 366).

X/Open Curses, Issue 7

349


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 ‘$’ and enclosed in angle brackets, as in `el=$\E$<3>`. The Curses implementation achieves delays by outputting to the terminal an appropriate number of system-defined padding characters. The `tpus()` function provides delays when used to send such a capability to the terminal.

The delay can be any of the following: a number, a number followed by an asterisk, such as 5*, a number followed by a slash, such as 5/, or a number followed by both, such as 5*/.

- A ‘**’ shows that the required delay is proportional to the number of lines affected by the operation, and the amount given is the delay required per affected unit. (In the case of insert characters, the factor is still the number of lines affected. This is always 1 unless the device has in and the software uses it.) When a ‘**’ is specified, it is sometimes useful to give a delay of the form 3.5 to specify a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.)

- A ‘/’ indicates that the delay is mandatory and padding characters are transmitted regardless of the setting of xon. If ‘/’ is not specified or if a device has xon defined, the delay information is advisory and is only used for cost estimates or when the device is in raw mode. However, any delay specified for bel or flash is treated as mandatory.

The following notation is valid in terminfo source files for specifying special characters:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Represents Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>^x</td>
<td>Control-x (for any appropriate x)</td>
</tr>
<tr>
<td>\a</td>
<td>Alert</td>
</tr>
<tr>
<td>\b</td>
<td>Backspace</td>
</tr>
<tr>
<td>\E or \e</td>
<td>An ESCAPE character</td>
</tr>
<tr>
<td>\f</td>
<td>Form feed</td>
</tr>
<tr>
<td>\l</td>
<td>Linefeed</td>
</tr>
<tr>
<td>\n</td>
<td>Newline</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage-return</td>
</tr>
<tr>
<td>\s</td>
<td>Space</td>
</tr>
<tr>
<td>\t</td>
<td>Tab</td>
</tr>
<tr>
<td>^</td>
<td>Caret (*)</td>
</tr>
<tr>
<td>\</td>
<td>Backslash ()</td>
</tr>
<tr>
<td>,</td>
<td>Comma (,)</td>
</tr>
<tr>
<td>:</td>
<td>Colon (:)</td>
</tr>
<tr>
<td>0</td>
<td>Null</td>
</tr>
<tr>
<td>\nnn</td>
<td>Any character, specified as three octal digits</td>
</tr>
</tbody>
</table>

(See the XBD specification, General Terminal Interface.)
Commented-out Capabilities

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the second `ind` in the example in Section 7.1.4 (on page 349). 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 \texttt{cols} numeric capability. If the device has a screen, then the number of lines on the screen is given by the \texttt{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 \texttt{am} capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the \texttt{clear} string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over) then it should have the \texttt{os} capability. If the device is a printing terminal, with no soft copy unit, specify both \texttt{hc} and \texttt{os}. If there is a way to move the cursor to the left edge of the current row, specify this as \texttt{cr}. (Normally this will be \texttt{<carriage-return>}, \texttt{<control-M>}.) If there is a way to produce an audible signal (such as a bell or a beep), specify it as \texttt{bel}. If, like most devices, the device uses the xon/xoff flow-control protocol, specify \texttt{xon}.

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

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

Parameterized versions of the scrolling sequences are \texttt{indn} and \texttt{rin}. These versions have the same semantics as \texttt{ind} and \texttt{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 \texttt{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 \texttt{cuf1} from the last column. Backward motion from the left edge of the screen is possible only when \texttt{bw} is specified. In this case, \texttt{cub1} will move to the right edge of the previous row. If \texttt{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, \texttt{am} should be specified in the \texttt{terminfo} source file. In this case, initialization strings should turn on this option, if possible. If the device has a command that moves to the first column of the next line, that command can be given as \texttt{nel} (newline). It does not matter if the command clears the remainder of the current line, so if the device has no \texttt{cr} and
If it may still be possible to craft a working ncl out of one or both of them.

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

5320|att5320|AT&T 5320 hardcopy terminal,
  am, hc, os,
cols#132,
bel=ˆG, cr=\r, cubi=\b, cnd1=\n,
dch1=\E[P, dll=\E[M,
ind=\n,

while the Lear Siegler ADM-3 is described as:

adm3 | lsi adm3,
am, bel=ˆG, clear=ˆZ, cols#80, cr=ˆM, cub1=ˆH,
cud1=ˆJ, ind=ˆJ, lines#24,

A.1.2 Parameterized Strings

Cursor addressing and other strings requiring arguments are described by a argumentized string capability with escapes in a form (%x) comparable to printf(). For example, to address the cursor, the cup capability is given, using two arguments: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory-relative cursor addressing, that can be indicated by mrcup.

The argument mechanism uses a stack and special % codes to manipulate the stack in the manner of Reverse Polish Notation (postfix). Typically, a sequence pushes one of the arguments onto the stack and then prints it in some format. Often more complex operations are necessary. Operations are in postfix form with the operands in the usual order. That is, to subtract 5 from the first argument, use %p1%{5}%−.

The % encodings have the following meanings:

%% Outputs ‘%’.
%[:flags]width[.precision]doxXs
  As in printf(); flags are [−+#] and space.
%c
  Print pop() gives %c.
%p[1-9]
  Push the ith argument.
%P[a-z]
  Set dynamic variable [a-z] to pop().
%g[a-z]
  Get dynamic variable [a-z] and push it.
%P[A-Z]
  Set static variable [a-z] to pop().
%g[A-Z]
  Get static variable [a-z] and push it.
%c
  Push char constant c.
%n
  Push decimal constant nn.
%l
  Push strlen(pop()).
%\m
  Arithmetic (%m is mod): push(pop \texttt{integer}_2 \texttt{op} pop \texttt{integer}_1) where \texttt{integer}_1 represents the top of the stack.
A.1.3 Cursor Motions

If the terminal has a fast way to home the cursor (to very upper-left corner of screen) then this can be given as home; similarly, a fast way of getting to the lower left-hand corner can be given as II; this may involve going up with cuu1 from the home position, but a program should never do this itself (unless II 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 \E& 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 \texttt{cup}. If there are argumentized local motions (such as “move \( n \) spaces to the right”), these can be given as \texttt{cud}, \texttt{cub}, \texttt{cuf}, and \texttt{cuu} with a single argument indicating how many spaces to move. These are primarily useful if the device does not have \texttt{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 \texttt{smcup} and \texttt{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 \texttt{smcup} sets the command character to be the one used by \texttt{terminfo}. If the \texttt{rmcup} will not restore the screen after an \texttt{smcup} sequence is output (to the state prior to outputting \texttt{smcup}) specify \texttt{nrmc}.

\subsection*{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 \texttt{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 \texttt{el1}. If the terminal can clear from the current position to the end of the display, then this should be given as \texttt{ed}. \texttt{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 \texttt{ed} is not available.)

\subsection*{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 \texttt{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 \texttt{dl1}; this is done only from the first position on the line to be deleted. Versions of \texttt{il1} and \texttt{dl1} which take a single argument and insert or delete that many lines can be given as \texttt{il} and \texttt{dl}.

If the terminal has a settable destructive scrolling region (like the VT100) the command to set this can be described with the \texttt{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 \texttt{sc} and \texttt{rc} (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using \texttt{ri} or \texttt{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 (\texttt{ri}) followed by a delete line (\texttt{dl1}) or index (\texttt{ind}). If the data that was originally on the bottom line of the scrolling region was restored into the scrolling region by \texttt{dl1} or \texttt{ind}, then the terminal has non-destructive scrolling regions. Otherwise, it has destructive scrolling regions. Do not specify \texttt{csr} if the terminal has non-destructive scrolling regions, unless \texttt{ind}, \texttt{ri}, \texttt{indn}, \texttt{rin}, \texttt{dl}, and \texttt{dl1} all simulate destructive scrolling.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the argumentized string \texttt{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 \texttt{da} capability should be given; if
display memory can be retained below, then \texttt{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 \texttt{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 \texttt{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 \texttt{<blank>}s on the screen, shifting upon an insert or delete only to an untyped \texttt{<blank>} on the screen which is either eliminated, or expanded to two untyped \texttt{<blank>}s. You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type \texttt{abc \ def} using local cursor motions (not spaces) between the \texttt{abc} and the \texttt{def}. Then position the cursor before the \texttt{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 \texttt{<blank>}s and untyped positions. If the \texttt{abc} shifts over to the \texttt{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 \texttt{in}, which stands for “insert null”. While these are two logically separate attributes (one line \textit{versus} multi-line insert mode, and special treatment of untyped spaces) we have seen no terminals whose insert mode cannot be described with the single attribute.

\texttt{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 \texttt{smir} the sequence to get into insert mode. Give as \texttt{rmir} the sequence to leave insert mode. Now give as \texttt{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 \texttt{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 \texttt{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 \texttt{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 \texttt{ip}.

If your terminal needs both to be placed into an “insert mode” and a special code to precede each inserted character, then both \texttt{smir/rmir} and \texttt{ich1} can be given, and both will be used. The \texttt{ich} capability, with one argument, \texttt{n}, will insert \texttt{n} \texttt{<blank>}s.

If padding is necessary between characters typed while not in insert mode, give this as a number of milliseconds padding in \texttt{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 \texttt{<tab>} after the insertion position). If your terminal allows motion while in insert mode you can give the capability \texttt{mir} to speed up inserting in this case. Omitting \texttt{mir} will affect only speed. Some terminals (notably Datamedia) must not have \texttt{mir} because of the way their insert mode works.

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

A command to erase \texttt{n} characters (equivalent to outputting \texttt{n} \texttt{<blank>}s without moving the cursor) can be given as \texttt{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.

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

Choose one display method as standout mode and use it to highlight error messages and other text to which you want to draw attention. Choose a form of display that provides strong contrast but that is easy on the eyes. (We recommend reverse-video plus half-bright or reverse-video alone.) The sequences to enter and exit standout mode are given as \texttt{smso} and \texttt{rmso}, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then \texttt{xmc} should be given to tell how many spaces are left.

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

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

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), then this can be given as \texttt{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 \texttt{cvvis}. The boolean \texttt{chts} should also be given. If there is a way to make the cursor completely invisible, give that as \texttt{civis}. The capability \texttt{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 \texttt{ul}. For devices on which a character overstriking another leaves both characters on the screen, specify the capability \texttt{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 tiparm() arguments, called here p1 through p9. 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>Argument</th>
<th>Attribute</th>
<th>Escape Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>standout</td>
<td>\E[0;4;7m</td>
</tr>
<tr>
<td>p2</td>
<td>underline</td>
<td>\E[0;3m</td>
</tr>
<tr>
<td>p3</td>
<td>reverse</td>
<td>\E[0;4m</td>
</tr>
<tr>
<td>p4</td>
<td>blink</td>
<td>\E[0;5m</td>
</tr>
<tr>
<td>p5</td>
<td>dim</td>
<td>\E[0;7m</td>
</tr>
<tr>
<td>p6</td>
<td>bold</td>
<td>\E[0;3;4m</td>
</tr>
<tr>
<td>p7</td>
<td>invis</td>
<td>\E[0;8m</td>
</tr>
<tr>
<td>p8</td>
<td>protect</td>
<td>not available</td>
</tr>
<tr>
<td>p9</td>
<td>altcharset</td>
<td>^O (off) ^N (on)</td>
</tr>
</tbody>
</table>

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

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

Now look at when different sequences are output. For example, ;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>terminfo 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;t;3%;</td>
</tr>
<tr>
<td>;4</td>
<td>if p1 or p3 or p6</td>
<td>%p1%p3%p6;t;4%;</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;t;7%;</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%;\^O%?%p9%\;t%N%e%O%;

Remember that sgr and sgr0 must always be specified.
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 smkx and rmkx. Otherwise, the keypad is assumed to always transmit.

The sequences sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as kcub1, kcuf1, kcuu1, kced1, and khome, respectively. If there are function keys such as f0, f1, ..., f63, the sequences they send can be specified as kf0, kf1, ..., kf63. If the first 11 keys have labels other than the default f0 through f10, the labels can be given as lfo, lfi, ..., lf10.

The codes transmitted by certain other special keys can be given: kll (home down), kbs (backspace), ktbc (clear all <tab>s), kctab (clear the tab stop in this column), kclr (clear screen or erase key), kdc1 (delete character), kdl1 (delete line), krmir (exit insert mode), kel (clear to end of line), ked (clear to end of screen), kich1 (insert character or enter insert mode), kil1 (insert line), knp (next page), kpp (previous page), kind (scroll forward/down), kri (scroll backward/up), khts (set a tab stop in this column). In addition, if the keypad has a 3-by-3 array of keys including the four arrow keys, the other five keys can be given as ka1, ka3, kb2, kc1, and kc3. These keys are useful when the effects of a 3-by-3 directional pad are needed. Further keys are defined above in the capabilities list.

Strings to program function keys can be specified as pfkey, pfloc, and pfx. A string to program screen labels should be specified as pln. Each of these strings takes two 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 pfx causes the string to be transmitted to the computer. The capabilities nlab, lw, and lh define the number of programmable screen labels and their width and height. If there are commands to turn the labels on and off, give them in pln sequences to make sure that the change becomes visible.

A.1.9 Tabs and Initialization

If the device has hardware tabs, the command to advance to the next tab stop can be given as ht (usually <control-I>). A “backtab” command that moves leftward to the next tab stop can be given as cbt. By convention, if tty modes show that <tab>s 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 <tab>s that are initially set every n spaces when the device is powered up, the numeric argument it is given, showing the number of spaces of the <tab>s 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 non-volatile memory, the terminfo description can assume that they are properly set. If there are commands to set and clear tab stops, they can be given as tbc (clear all tab stops) and hts (set a tab stop in the current column of every row).

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

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

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

Any margin can be cleared with mgc. (For instructions on how to specify commands to set and
clear margins, see Margins (on page 371).)

A.1.10 Delays

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

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 hs should be given. Special strings that go to a
given column of the status line and return from the status line can be given as tsl and fsl. (fsl
must leave the cursor position in the same place it was before tsl. If necessary, the sc and rc
strings can be included in tsl and fsl to get this effect.) The capability tsl takes one argument,
which is the column number of the status line the cursor is to be moved to.

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

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 acsc. The definition of this string is based on the alternate character set used in the
Digital VT100 terminal, extended slightly with some characters from the AT&T 4410v1 terminal.
### Glyph Names and VT100+ Characters

<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>//</td>
</tr>
<tr>
<td>arrow pointing up</td>
<td>-</td>
</tr>
<tr>
<td>diamond</td>
<td>\</td>
</tr>
<tr>
<td>checker board (stipple)</td>
<td>a</td>
</tr>
<tr>
<td>degree symbol</td>
<td>f</td>
</tr>
<tr>
<td>plus/minus</td>
<td>g</td>
</tr>
<tr>
<td>board of squares</td>
<td>h</td>
</tr>
<tr>
<td>lower right corner</td>
<td>j</td>
</tr>
<tr>
<td>upper right corner</td>
<td>k</td>
</tr>
<tr>
<td>upper left corner</td>
<td>l</td>
</tr>
<tr>
<td>lower left corner</td>
<td>m</td>
</tr>
<tr>
<td>plus</td>
<td>n</td>
</tr>
<tr>
<td>scan line 1</td>
<td>o</td>
</tr>
<tr>
<td>horizontal line</td>
<td>q</td>
</tr>
<tr>
<td>scan line 9</td>
<td>s</td>
</tr>
<tr>
<td>left tee (├)</td>
<td>t</td>
</tr>
<tr>
<td>right tee (┼)</td>
<td>u</td>
</tr>
<tr>
<td>bottom tee (┘)</td>
<td>v</td>
</tr>
<tr>
<td>top tee (┌)</td>
<td>w</td>
</tr>
<tr>
<td>vertical line</td>
<td>x</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:

```
acsc=1RmFKTjGq\,x.
```

In addition, `terminfo` lets you define multiple character sets (see Section A.2.5, on page 373).
A.1.13 Color Manipulation

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

- **Tektronix-style**
  
  The Tektronix method uses a set of \( N \) predefined colors (usually 8) from which an application can select “current” foreground and background colors. Thus, a terminal can support up to \( N \) colors mixed into \( N^2 \) color-pairs to be displayed on the screen at the same time.

- **Hewlett-Packard-style**
  
  In the HP method, the application cannot define the foreground independently of the background, or *vice versa*. Instead, the application must define an entire color-pair at once. Up to \( M \) color-pairs, made from \( 2^M \) different colors, can be defined this way.

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

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

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

If the terminal supports ANSI escape sequences to set background and foreground, they should be coded as **setab** and **setaf**, respectively. If the terminal supports other escape sequences to set background and foreground, they should be coded as **setb** and **setf**, respectively. The **vidputs()** function and the refresh functions use **setab** and **setaf** if they are defined. Each of these capabilities requires one argument: the number of the color. By convention, the first eight colors (0–7) map to, in order: black, red, green, yellow, blue, magenta, cyan, white. However, color re-mapping may occur or the underlying hardware may not support these colors. Mappings for any additional colors supported by the device (that is, to numbers greater than 7) are at the discretion of the **terminfo** entry writer.

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

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

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Bit Position</th>
<th>Decimal Value</th>
<th>Characteristic That Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA_STANDOUT</td>
<td>0</td>
<td>1</td>
<td>sgr, parameter 1</td>
</tr>
<tr>
<td>WA_UNDERLINE</td>
<td>1</td>
<td>2</td>
<td>sgr, parameter 2</td>
</tr>
<tr>
<td>WA_REVERSE</td>
<td>2</td>
<td>4</td>
<td>sgr, parameter 3</td>
</tr>
<tr>
<td>WA_BLINK</td>
<td>3</td>
<td>8</td>
<td>sgr, parameter 4</td>
</tr>
<tr>
<td>WA_DIM</td>
<td>4</td>
<td>16</td>
<td>sgr, parameter 5</td>
</tr>
<tr>
<td>WA_BOLD</td>
<td>5</td>
<td>32</td>
<td>sgr, parameter 6</td>
</tr>
<tr>
<td>WA_INVIS</td>
<td>6</td>
<td>64</td>
<td>sgr, parameter 7</td>
</tr>
<tr>
<td>WA_PROTECT</td>
<td>7</td>
<td>128</td>
<td>sgr, parameter 8</td>
</tr>
<tr>
<td>WA_ALTCARSET</td>
<td>8</td>
<td>256</td>
<td>sgr, parameter 9</td>
</tr>
<tr>
<td>WA_HORIZONTAL</td>
<td>9</td>
<td>512</td>
<td>sgr1, parameter 1</td>
</tr>
<tr>
<td>WA_LEFT</td>
<td>10</td>
<td>1024</td>
<td>sgr1, parameter 2</td>
</tr>
<tr>
<td>WA_LOW</td>
<td>11</td>
<td>2048</td>
<td>sgr1, parameter 3</td>
</tr>
<tr>
<td>WA_RIGHT</td>
<td>12</td>
<td>4096</td>
<td>sgr1, parameter 4</td>
</tr>
<tr>
<td>WA_TOP</td>
<td>13</td>
<td>8192</td>
<td>sgr1, parameter 5</td>
</tr>
<tr>
<td>WA_VERTICAL</td>
<td>14</td>
<td>16384</td>
<td>sgr1, parameter 6</td>
</tr>
</tbody>
</table>

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

A.1.14 Miscellaneous

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

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

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters) this can be indicated with the argumentized string rep. The first argument is the character to be repeated and the second is the number of times to repeat it. Thus:

```
tiparm(repeat_char, 'x', 10)
```

is the same as xxxxxxxxxxx.

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

Terminal descriptions that do not represent a specific kind of known terminal, such as switch,
dialup, patch, and network, should include the gn (generic) capability so that programs can
complain that they do not know how to talk to the terminal. (This capability does not apply to
virtual terminal descriptions for which the escape sequences are known.) If the terminal is one of
those supported by the virtual terminal protocol, the terminal number can be given as vt. A
line-turn-around sequence to be transmitted before doing reads should be specified in rfi.

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

If the terminal has a “meta key” which acts as a shift key, setting the eighth bit of any character
transmitted, this fact can be indicated with km. Otherwise, software will assume that the eighth
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 <tab>s turn all characters moved over to <blank>s, should
indicate xt (destructive <tab>s). 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 7.1.3 (on page 340).

#### A.2.1 Rounding Values

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

#### 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</th>
<th>Number of Smallest Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>orhi</td>
<td>Steps per inch horizontally</td>
</tr>
<tr>
<td>orvi</td>
<td>Steps per inch vertically</td>
</tr>
<tr>
<td>orc</td>
<td>Steps per column</td>
</tr>
<tr>
<td>orl</td>
<td>Steps per line</td>
</tr>
</tbody>
</table>

When printing in normal mode, each character printed causes movement to the next column, except in special cases described later; the distance moved is the same as the per-column resolution. Some printers cause an automatic movement to the next line when a character is printed in the rightmost position; the distance moved vertically is the same as the per-line resolution. When printing in micro mode, these distances can be different, and may be zero for some printers.

<table>
<thead>
<tr>
<th>Automatic Motion after Printing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Normal Mode:</strong></td>
</tr>
<tr>
<td>ort</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.
Automatic Motion after Printing Wide Character

<table>
<thead>
<tr>
<th>Normal Mode or Micro Mode (mcs=orc): widcs</th>
<th>Steps moved horizontally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Micro Mode (mcs&lt;orc): mcs</td>
<td>Steps moved horizontally</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>Changing the Character/Line Pitches</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpi</td>
</tr>
<tr>
<td>cpix</td>
</tr>
<tr>
<td>lpi</td>
</tr>
<tr>
<td>lpix</td>
</tr>
<tr>
<td>chr</td>
</tr>
<tr>
<td>cvr</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 375).

<table>
<thead>
<tr>
<th>Effects of Changing the Character/Line Pitches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before</td>
</tr>
<tr>
<td>Using cpi with cpix clear:</td>
</tr>
<tr>
<td>orhi'</td>
</tr>
<tr>
<td>orc'</td>
</tr>
<tr>
<td>Using cpi with cpix set:</td>
</tr>
<tr>
<td>orhi'</td>
</tr>
<tr>
<td>orc'</td>
</tr>
<tr>
<td>Using lpi with lpix clear:</td>
</tr>
<tr>
<td>orvi'</td>
</tr>
<tr>
<td>orl'</td>
</tr>
<tr>
<td>Using lpi with lpix set:</td>
</tr>
<tr>
<td>orvi'</td>
</tr>
<tr>
<td>orl'</td>
</tr>
<tr>
<td>Using chr:</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 smaller 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>mcf1</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>mcfu</td>
<td>Move N steps right</td>
</tr>
<tr>
<td>mceu</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>xvpa</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>mjum</td>
<td>Limit on use of mcub1, mcfu1, mcuu1, mcud1</td>
</tr>
<tr>
<td>maddr</td>
<td>Limit on use of mhpa, xvpa</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 xvpa can’t move up</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Entering/Exiting Micro Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>smicm</td>
</tr>
<tr>
<td>rmicm</td>
</tr>
<tr>
<td>crxm</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>What Happens After Character Printed in Rightmost Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>sam</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Entering/Exiting Reverse Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>slm</td>
</tr>
<tr>
<td>rlm</td>
</tr>
<tr>
<td>sum</td>
</tr>
<tr>
<td>rum</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>While sense of horizontal motions reversed:</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcub1</td>
</tr>
<tr>
<td>mcuf1</td>
</tr>
<tr>
<td>mcub</td>
</tr>
<tr>
<td>mcuf</td>
</tr>
<tr>
<td>cub1</td>
</tr>
<tr>
<td>cuff1</td>
</tr>
<tr>
<td>cub</td>
</tr>
<tr>
<td>cuf</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>While sense of vertical motions reversed:</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcuu1</td>
</tr>
<tr>
<td>mcud1</td>
</tr>
<tr>
<td>mcuu</td>
</tr>
<tr>
<td>mcud</td>
</tr>
<tr>
<td>cuu1</td>
</tr>
<tr>
<td>cud1</td>
</tr>
<tr>
<td>cuu</td>
</tr>
<tr>
<td>cud</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.

<table>
<thead>
<tr>
<th>Miscellaneous Motion Strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>docr</td>
</tr>
<tr>
<td>zeron</td>
</tr>
</tbody>
</table>

Margins

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

<table>
<thead>
<tr>
<th>Setting Margins</th>
</tr>
</thead>
<tbody>
<tr>
<td>smgl</td>
</tr>
<tr>
<td>smgr</td>
</tr>
<tr>
<td>smgb</td>
</tr>
<tr>
<td>smgt</td>
</tr>
<tr>
<td>smgbp</td>
</tr>
<tr>
<td>smglp</td>
</tr>
<tr>
<td>smgrp</td>
</tr>
<tr>
<td>smgtp</td>
</tr>
</tbody>
</table>

The last four strings are used with one or more arguments that give the position of the margin or margins to set. If both of smglp and smgrp are set, each is used with a single argument, N, that gives the column number of the left and right margin, respectively. If both of smgt and smgb are set, each is used to set the top and bottom margin, respectively: smgt is used with a single argument, N, the line number of the top margin; however, smgb is used with two arguments, N and MM, 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 smgt and smgb 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 smgl and smgrp or smgt and smgb should be defined; the other should be left blank. When writing an application that uses these string capabilities, the pairs should be first checked to see if each in the pair is set or only one is set, and should then be used accordingly.

In counting lines or columns, line zero is the top line and column zero is the left-most column. A zero value for the second argument with smgbp means the bottom line of the page.
All margins can be cleared with \texttt{mgc}.

**Shadows, Italics, Wide Characters, Superscripts, Subscripts**

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

<table>
<thead>
<tr>
<th>Enhanced Printing</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sshm</td>
<td>Enter shadow-printing mode</td>
</tr>
<tr>
<td>rshm</td>
<td>Exit shadow-printing mode</td>
</tr>
<tr>
<td>sitm</td>
<td>Enter italicizing mode</td>
</tr>
<tr>
<td>ritm</td>
<td>Exit italicizing mode</td>
</tr>
<tr>
<td>swidm</td>
<td>Enter wide character mode</td>
</tr>
<tr>
<td>rwidm</td>
<td>Exit wide character mode</td>
</tr>
<tr>
<td>ssupm</td>
<td>Enter superscript mode</td>
</tr>
<tr>
<td>rsupm</td>
<td>Exit superscript mode</td>
</tr>
<tr>
<td>supcs</td>
<td>List of characters available as superscripts</td>
</tr>
<tr>
<td>ssubm</td>
<td>Enter subscript mode</td>
</tr>
<tr>
<td>rsubm</td>
<td>Exit subscript mode</td>
</tr>
<tr>
<td>subcs</td>
<td>List of characters available as subscripts</td>
</tr>
</tbody>
</table>

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

\texttt{terminfo} also has a capability for printing emboldened text (\texttt{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 involves overstriking, but with a slight movement up and/or to the side so that the character is “fatter”.

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

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

If only a subset of the printable ASCII characters can be printed as superscripts or subscripts, they should be listed in \texttt{supcs} or \texttt{subcs} strings, respectively. If the \texttt{ssupm} or \texttt{ssubm} strings contain control sequences, but the corresponding \texttt{supcs} or \texttt{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:

\[
B_i \quad B_i \quad B^i
\]

Note that the existing \texttt{msg} 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. \texttt{msg} should be set for those printers that accept any motion control sequences without affecting shadow, italicized, widened, superscript, or subscript printing. Conversely, if \texttt{msg} 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 361), terminfo lets you define alternate character sets. The following capabilities cover printers and terminals with multiple selectable or definable character sets:

<table>
<thead>
<tr>
<th>Alternate Character Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>scs</td>
</tr>
<tr>
<td>scsd</td>
</tr>
<tr>
<td>defc</td>
</tr>
<tr>
<td>rcsd</td>
</tr>
<tr>
<td>csnm</td>
</tr>
<tr>
<td>daisy</td>
</tr>
</tbody>
</table>

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

Character set 0 is the default character set present after the printer has been initialized. Not every printer has 64 character sets, of course; using scs with an argument that doesn’t select an available character set should cause a null pointer to be returned by tiparm().

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 tiparm() 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 tiparm(), 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 lowercase 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 374).

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

When used with a character set number in a call to tiparm(), the csnm string will produce the equivalent name. These names should be used as a reference only. No naming convention is implied, although anyone who creates a terminfo entry for a printer should use names consistent with the names found in user documents for the printer. Application developers should allow a user to specify a character set by number (leaving it up to the user to examine the csnm string to determine the correct number), or by name, where the application examines the csnm string to determine the corresponding character set number.
These capabilities are likely to be used only with dot-matrix printers. If they are not available, the strings should not be defined. For printers that have manually changed print-wheels or font cartridges, the boolean `daisy` is set.

### 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.

<table>
<thead>
<tr>
<th>Dot-Matrix Graphics</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>npins</code></td>
</tr>
<tr>
<td><code>spinv</code></td>
</tr>
<tr>
<td><code>spinh</code></td>
</tr>
<tr>
<td><code>porder</code></td>
</tr>
<tr>
<td><code>sbim</code></td>
</tr>
<tr>
<td><code>rbim</code></td>
</tr>
</tbody>
</table>

The `sbim` string is used with a single argument, \( B \), the width of the image in dots.

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

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

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

An application computes the “image-data bytes” from the internal image, mapping vertical dot positions in each print-head pass into 8-bit bytes, using a 1 bit where ink should be applied and 0 where no ink should be applied. This can be reversed (0 bit for ink, 1 bit for no ink) by giving a negative pin number. If a position is skipped in `porder`, a 0 bit is used. If a position has a lowercase ‘\( \times \)’ instead of a pin number, a 1 bit is used in the skipped position. For consistency, a lowercase ‘\( \circ \)’ can be used to represent a 0 filled, skipped bit. There must be a multiple of 8-bit positions used or skipped in `porder`; if not, low-order bits of the last byte are set to 0. The offset, if given, is added to each data byte; the offset can be negative.

Some examples may help clarify the use of the `porder` string. The AT&T 470, AT&T 475, and
C. Itoh 8510 printers provide eight pins for graphics. The pins are identified top to bottom by the 8 bits in a byte, from least significant to most. The `porder` strings for these printers would be \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 `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 `porder` string for these printers would be \texttt{,,6,5,4,3,2,1;63}, or alternately \texttt{o,0,6,5,4,3,2,1;63}.

### 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:

<table>
<thead>
<tr>
<th>Changing the Character/Line Pitches</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cpi</code></td>
<td>Change character pitch</td>
<td><code>spinh</code></td>
</tr>
<tr>
<td><code>cpix</code></td>
<td>If set, <code>cpi</code> changes <code>spinh</code></td>
<td></td>
</tr>
<tr>
<td><code>lpi</code></td>
<td>Change line pitch</td>
<td><code>spinv</code></td>
</tr>
<tr>
<td><code>lpix</code></td>
<td>If set, <code>lpi</code> changes <code>spinv</code></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Effects of Changing the Character/Line Pitches</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using <code>cpi</code> with <code>cpix</code> clear:</td>
<td><code>spinh′</code></td>
<td><code>spinh</code></td>
</tr>
<tr>
<td>Using <code>cpi</code> with <code>cpix</code> set:</td>
<td><code>spinh′</code></td>
<td><code>spinv = spinh′ · \frac{orhi}{orhi′}</code></td>
</tr>
<tr>
<td>Using <code>lpi</code> with <code>lpix</code> clear:</td>
<td><code>spinv′</code></td>
<td><code>spinv</code></td>
</tr>
<tr>
<td>Using <code>lpi</code> with <code>lpix</code> set:</td>
<td><code>spinv′</code></td>
<td><code>spinv = spinv′ · \frac{orhi}{orhi′}</code></td>
</tr>
<tr>
<td>Using <code>chr</code>:</td>
<td><code>spinh′</code></td>
<td><code>spinh</code></td>
</tr>
<tr>
<td>Using <code>cvr</code>:</td>
<td><code>spinv′</code></td>
<td><code>spinv</code></td>
</tr>
</tbody>
</table>

`orhi′` and `orhi` are the values of the horizontal resolution in steps per inch, before using `cpi` and after using `cpi`, respectively. Likewise, `orvi′` and `orv` are the values of the vertical resolution in steps per inch, before using `lpi` and after using `lpi`, respectively. Thus, the changes in the dots per inch for dot-matrix graphics follow the changes in steps per inch for printer resolution.
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 synchronized with a printer, and because modern printers can buffer data before printing it, a program generally cannot determine at any time what has been printed. Two numeric capabilities can help a program estimate what has been printed.

<table>
<thead>
<tr>
<th>Print Rate/Buffer Size</th>
</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 synchronize itself with the printer.

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

Applications that use these values should recognize the variability in the print rate. Straight text, in short lines, with no embedded control sequences will probably print at close to the advertised print rate and probably faster than the rate in cps. Graphics data with a lot of control sequences, or very long lines of text, will print at well below the advertised rate and below the rate in cps. If the application is using cps to decide how long it should take a printer to print a block of text, the application should pad the estimate. If the application is using cps to decide how much text has already been printed, it should shrink the estimate. The application will thus err in favor of the user, who wants, above all, to see all the output in its correct place.
A.3 Selecting a Terminal

If the environment variable \texttt{TERMINFO} is defined, any program using Curses checks for a local terminal definition before checking in the standard place. For example, on implementations which use the traditional directory layout for the \texttt{terminfo} data, if \texttt{TERM} is set to \texttt{att4424}, then the compiled terminal definition is found in by default the path:

\texttt{a/att4424}

within an implementation-specific directory.

(The a is copied from the first letter of \texttt{att4424} to avoid creation of huge directories.) However, if \texttt{TERMINFO} is set to \texttt{$HOME/myterms}, Curses first checks:

\texttt{$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 \texttt{LINES} and \texttt{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 \texttt{terminfo}. The \texttt{use_env()} function can be used to override this default behavior.

A.4 Application Usage

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in \texttt{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 \texttt{TERMINFO} can be set to the pathname of a directory containing the compiled description, and programs will look there rather than in the \texttt{terminfo} database.

A.4.1 Conventions for Device Aliases

Every device must be assigned a name, such as \texttt{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 \texttt{−w} suffix means \textit{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 \texttt{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 optimizations 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 17).

Curses window
Data structures, which can be thought of as two-dimensional arrays of characters that represent screen displays. These data structures are manipulated with Curses functions.

cursor position
The line and column position on the screen denoted by the terminal’s cursor.

empty wide-character string
A wide-character string whose first element is a null wide-character code.

erase character
A special input character that deletes the last character in the current line, if there is one.

kill character
A special input character that deletes all data in the current line, if there are any.

null chtype
A chtype with all bits set to zero.

null wide-character code
A wide-character code with all bits set to zero.

pad
A window that is not necessarily associated with a viewable part of a screen.

parent window
A window that has subwindows or derived windows associated with it.

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

SCREEN
An opaque Curses data type that is associated with the display screen.

subwindow
A window, created within another window, but positioned relative to that other window. Changes made to a subwindow do not affect its parent window. A derived window differs from a subwindow only in that it is positioned relative to the origin of its parent window. Changes to a parent window will affect both subwindows and derived windows.

touch
To set a flag in a window that indicates that the information in the window could differ from 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 ................................................................. 377
<curses.h> .............................................................. 306
<term.h> ............................................................... 320
<unctrl.h> ............................................................. 321
@ .............................................................................. 366

XBD specification
relationship to ....................................................... 13

XSH specification
relationship to ....................................................... 13

_w infix ................................................................. 25
_XOPEN_SOURCE ...................................................... 10

add
   effect on straddling character .......................... 20
   resulting rendition .......................................... 21
add function ......................................................... 18
addch() ................................................................. 36
addchnstr............................................................ 37
addchstr() ............................................................ 37
addnstr() ............................................................. 38
addnwstr() .......................................................... 40
addstr ................................................................. 38
addwstr ............................................................... 40
add_wch() ............................................................ 33
add_wchnstr........................................................ 34
add_wchstr .......................................................... 34

adjustment of cursor position .............................. 19
adverted print rate ................................................. 376

advisory delay .........................................................

alias
   in terminfo ........................................................ 337
alternate character set ........................................ 358, 373
   line drawing ...................................................... 361
alternate keypad .................................................. 360
am ................................................................. 353
   ignoring linefeed after ...................................... 365
ancestor ............................................................... 15
Ann Arbor 4080 (example) ................................... 355
ANSI foreground/background ................................ 363
application consideration ..................................... 27
area clear .............................................................. 356
arrow keys .......................................................... 360
asterisk
   in terminfo ........................................................ 350
AT&T 4410v1
   line drawing .................................................... 361
AT&T 470/475 ....................................................... 374
Index

AT&T 5320 (example) ................................................................. 354
AT&T 610 (example) ................................................................. 349
attribute ................................................................................. 16
attroff() .................................................................................. 41
attron ................................................................. 358
attrset ................................................................. 43
attr_get() ................................................................................. 43
attr_off ................................................................................. 41
attr_on ................................................................................. 41
attr_set ................................................................................. 41
audible signal................................................................. 353
automatic margin................................................................. 353
automatic motion ................................................................. 372
auxiliary printer control .......................................................... 365
background ........................................................................ 18, 379
background character ............................................................. 18
   implicit use ........................................................................ 21
background color ................................................................. 363
backslash
   use in terminfo ................................................................. 337
backslash in terminfo ............................................................. 350
backspace
   special processing ................................................................. 21
basic capability ................................................................. 353
baud rate, versus printer throughput ..................................... 376
baudrate() ............................................................................. 44
bce ................................................................. 363
Beehive Superbee ................................................................. 366
beep() ................................................................................ 45
bel ...................................................................................... 353
delays .................................................................................. 350
bell ...................................................................................... 353
   visible ...................................................................... 358
bidirectional writing ............................................................. 3
bgkd() .................................................................................. 46
bkgdset ................................................................................. 46
bkgnd() .............................................................................. 48
bkgndset .............................................................................. 48
blanking text ........................................................................ 358
blink ................................................................. 358
blinking screen ................................................................. 358
block cursor ................................................................. 358
block mode ........................................................................ 27
bold ................................................................................... 358
   printing .................................................................. 372
boolean capability ................................................................. 338
border() .............................................................................. 50
   eliminates straddling characters ..................................... 20
border_set() ....................................................................... 52
box drawing ........................................................................ 353
box() .............................................................................. 54
box_set() ........................................................................... 55
brightness of character ........................................................................................................... 358
buffer size ................................................................................................................................. 376
bufsz ........................................................................................................................................... 376
bw .................................................................................................................................................. 353
C.Itoh 8510 ................................................................................................................................. 374
calculating print rate .................................................................................................................. 376
can .................................................................................................................................................. 4
can_change_color() ..................................................................................................................... 56
capability of device ................................................................................................................... 337
capability, device ......................................................................................................................... 338
carriage-return
  special processing ..................................................................................................................... 21
cbreak() ......................................................................................................................................... 59
cbt .................................................................................................................................................... 360
CC environment variable ........................................................................................................... 364
ccc .................................................................................................................................................... 363
change
  affecting subwindow .................................................................................................................. 14
change resolution ........................................................................................................................ 375
carriage
  replacement ............................................................................................................................... 18
  resulting rendition ....................................................................................................................... 21
  straddling .................................................................................................................................... 20
carriage-insert/delete ................................................................................................................ 19, 357
carriage-set
  alternate ...................................................................................................................................... 358, 373
  as sub/superscript ..................................................................................................................... 372
  line drawing ............................................................................................................................... 361
  name ............................................................................................................................................ 373
carriage-spacing
  character ....................................................................................................................................... 366
  chgat() ......................................................................................................................................... 60
  chr .................................................................................................................................................... 368
  recalculate resolution after ....................................................................................................... 368
  chts ............................................................................................................................................... 358
civis ............................................................................................................................................... 358
clear ............................................................................................................................................... 353
clear-screen .................................................................................................................................. 353
clear to end-of-line ...................................................................................................................... 356
clear() ........................................................................................................................................... 61
clearok() ....................................................................................................................................... 62
clipping of window ..................................................................................................................... 14
clrbot() .......................................................................................................................................... 64
clrtteol() ....................................................................................................................................... 65
cmdch ............................................................................................................................................ 364
cnorm ............................................................................................................................................ 358
codeset .......................................................................................................................................... 1
color ............................................................................................................................................. 16
color-background ........................................................................................................................ 363
COLORS .......................................................................................................................................... 30
colors ............................................................................................................................................ 363
color_content .................................................................................................................................. 56
color_content() ............................................................................................................................ 66

X/Open Curses, Issue 7  383
COLOR_PAIRES ........................................................................................................................................ 30
color_set .................................................................................................................................................. 41
color_set() ........................................................................................................................................... 67
COLS ..................................................................................................................................................... 31
cols ....................................................................................................................................................... 353
  status line ........................................................................................................................................ 361
column
  orphaned ........................................................................................................................................... 18
COLUMNS ................................................................................................................................................ 377
comma
  after last entry in terminfo .............................................................................................................. 349
  use in terminfo .................................................................................................................................. 337
command character .............................................................................................................................. 364
comment in terminfo ........................................................................................................................... 351
compilation environment ...................................................................................................................... 10
complex character
  function naming ................................................................................................................................... 25
Concept (example) ........................................................................................................................................ 356
Concept 100
  ignoring linefeed after wrap ............................................................................................................ 365
Concept 100 (example) ........................................................................................................................... 357
conformance .......................................................................................................................................... 3
conventions, lexical ............................................................................................................................... 339
cookie .................................................................................................................................................... 358
coordinate pair ...................................................................................................................................... 35
  after last entry in terminfo .............................................................................................................. 349
  use in terminfo .................................................................................................................................. 337
column
  orphaned ........................................................................................................................................... 18
copywin() ................................................................................................................................................. 68
cpi ........................................................................................................................................................... 375
  recalculate resolution after .............................................................................................................. 368
cpix ......................................................................................................................................................... 375
cpi[x] .................................................................................................................................................... 368
cps ......................................................................................................................................................... 376
cr ............................................................................................................................................................ 353
delays .................................................................................................................................................... 361
crxm ......................................................................................................................................................... 370
csnm ....................................................................................................................................................... 373
csr ......................................................................................................................................................... 356
cub ......................................................................................................................................................... 355
cub1 ....................................................................................................................................................... 355
  delays ................................................................................................................................................. 361
cud ......................................................................................................................................................... 355
cuf ......................................................................................................................................................... 353
cuf1 ....................................................................................................................................................... 353
cup ......................................................................................................................................................... 354
current or specified position .................................................................................................................. 25
current or specified window .................................................................................................................. 24
current position .................................................................................................................................... 369
curscr ..................................................................................................................................................... 71
Curses ....................................................................................................................................................... 1
Curses window ....................................................................................................................................... 379
cursor
  actual position ................................................................................................................................. 18
  analogue in printing terminal ........................................................................................................... 369
Index

appearance of ............................................................................................................................................................ 358

cursor addressing .......................................................................................................................................................... 354

cursor movement .......................................................................................................................................................... 353

  relocation .......................................................................................................................................................... 19

  within row or column ............................................................................................................................................... 355

cursor position .................................................................................................................................................. 18, 379

  at insert/delete .................................................................................................................................................. 19

curs_set() .......................................................................................................................................................... 70

cur_term .......................................................................................................................................................... 69

cuu .......................................................................................................................................................... 355

cuu1 .......................................................................................................................................................... 355

cu[b/d/f/u][1] .................................................................................................................................................. 370

cvr .......................................................................................................................................................... 368

  recalculate resolution after .................................................................................................................................. 368

cvvis .......................................................................................................................................................... 358

da .......................................................................................................................................................... 356
daaisy .......................................................................................................................................................... 373
darkened printing .................................................................................................................................................. 372
data types .................................................................................................................................................. 12
database, terminfo .................................................................................................................................................. 337

Datamedia (example) .................................................................................................................................................. 357
db .......................................................................................................................................................... 356
dch .......................................................................................................................................................... 357
dch1 .......................................................................................................................................................... 357
defc .......................................................................................................................................................... 373
definition, sharing .................................................................................................................................................. 366
def_prog_mode() .................................................................................................................................................. 72
def_shell_mode .................................................................................................................................................. 72
delay .......................................................................................................................................................... 350, 361
delay mode .................................................................................................................................................. 24
delay_output() .................................................................................................................................................. 75
delch() .......................................................................................................................................................... 76
delete

  effect on straddling character .................................................................................................................................. 20

delete/insert character .................................................................................................................................................. 357

delete/insert line .................................................................................................................................................. 356

deleteln() .......................................................................................................................................................... 77
deletion .......................................................................................................................................................... 20
delscreen() .......................................................................................................................................................... 78
delwin() .......................................................................................................................................................... 79
del_curterm() .................................................................................................................................................. 73
derived window .................................................................................................................................................. 15
derwin() .......................................................................................................................................................... 80
description of device .................................................................................................................................................. 337
destructive scrolling .................................................................................................................................................. 356
destructive tab .................................................................................................................................................. 365
device capability .................................................................................................................................................. 337
device name .................................................................................................................................................. 377
dialup terminal .................................................................................................................................................. 364

Digital LA100, LN03 .................................................................................................................................................. 374
dim .......................................................................................................................................................... 358
direct cursor addressing .................................................................................................................................................. 354
Index

dl.......................................................................................................................... 356
dl1......................................................................................................................... 356
docr ...................................................................................................................... 371
dot-matrix graphics ......................................................................................... 374
doupdate() ....................................................................................................... 82
draft-quality ................................................................................................. 376
drawing a box ............................................................................................... 353
dsl ..................................................................................................................... 361
dupwin() ......................................................................................................... 83
EC ...................................................................................................................... 5
ech ..................................................................................................................... 357
echo processing ............................................................................................ 24
echo( ) ........................................................................................................... 84
echochar() ..................................................................................................... 86
echo wchar() .................................................................................................. 85
ed ..................................................................................................................... 356
eighth bit ........................................................................................................ 365
e1 ..................................................................................................................... 356
e11 ................................................................................................................... 356
empty wide-character string ........................................................................ 379
emulator, terminal ....................................................................................... 363
en .................................................................................................................... 355
enacs .............................................................................................................. 358
end-of-line
  truncation/wrapping .................................................................................. 19
endwin() ........................................................................................................ 87
enhanced character set ................................................................................ 2
enhancement, turn off .................................................................................. 358
eo .................................................................................................................. 358
erase ............................................................................................................... 61
erase character ............................................................................................. 379
erase to end-of-line ..................................................................................... 356
erase( ) ......................................................................................................... 88
erasechar() ................................................................................................... 89
erasewchar ................................................................................................... 89
error numbers ............................................................................................ 13
escape in terminfo ...................................................................................... 350
escape sequence ......................................................................................... 14
eslok .............................................................................................................. 361
estimating printer throughput ..................................................................... 376
et ................................................................................................................... 355
extension
  EC .................................................................................................................. 5
  OB .................................................................................................................. 5
extra line of screen ....................................................................................... 361
extra-bright character ................................................................................ 358
ff ................................................................................................................... 364
delays ............................................................................................................. 361
filter() .......................................................................................................... 91
first line in terminfo .................................................................................... 337
flag, touched ............................................................................................... 14
flash .............................................................................................................. 358
**Index**

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>delays</td>
<td>350</td>
</tr>
<tr>
<td>flash()</td>
<td>92</td>
</tr>
<tr>
<td>flashing screen</td>
<td>358</td>
</tr>
<tr>
<td>flow control</td>
<td>365</td>
</tr>
<tr>
<td>flushin()</td>
<td>93</td>
</tr>
<tr>
<td>foreground color</td>
<td>363</td>
</tr>
<tr>
<td>form feed</td>
<td>371</td>
</tr>
<tr>
<td>format of entries</td>
<td>6</td>
</tr>
<tr>
<td>format of terminfo</td>
<td>337</td>
</tr>
<tr>
<td>fsl</td>
<td>361</td>
</tr>
<tr>
<td>function naming</td>
<td>24</td>
</tr>
<tr>
<td>functions</td>
<td></td>
</tr>
<tr>
<td>implementation</td>
<td>9</td>
</tr>
<tr>
<td>use</td>
<td>9</td>
</tr>
<tr>
<td>generic terminal description</td>
<td>364</td>
</tr>
<tr>
<td>getbegyx()</td>
<td>97</td>
</tr>
<tr>
<td>getbkgd</td>
<td>46</td>
</tr>
<tr>
<td>getbkgd()</td>
<td>99</td>
</tr>
<tr>
<td>getbkgrnd</td>
<td>48</td>
</tr>
<tr>
<td>getbkgrnd()</td>
<td>100</td>
</tr>
<tr>
<td>getchar()</td>
<td>101</td>
</tr>
<tr>
<td>getch()</td>
<td>102</td>
</tr>
<tr>
<td>getmaxyx()</td>
<td>97</td>
</tr>
<tr>
<td>getmaxyx()</td>
<td>104</td>
</tr>
<tr>
<td>getnstr()</td>
<td>107</td>
</tr>
<tr>
<td>getn_wstr()</td>
<td>105</td>
</tr>
<tr>
<td>getparyx()</td>
<td>97</td>
</tr>
<tr>
<td>getparyx()</td>
<td>109</td>
</tr>
<tr>
<td>getstr()</td>
<td>107</td>
</tr>
<tr>
<td>getstr()</td>
<td>110</td>
</tr>
<tr>
<td>getwin()</td>
<td>111</td>
</tr>
<tr>
<td>gety()</td>
<td>97</td>
</tr>
<tr>
<td>getyx()</td>
<td>112</td>
</tr>
<tr>
<td>get_wch()</td>
<td>94</td>
</tr>
<tr>
<td>get_wstr()</td>
<td>105</td>
</tr>
<tr>
<td>get_wstr()</td>
<td>96</td>
</tr>
<tr>
<td>glitch, magic cookie</td>
<td>358</td>
</tr>
<tr>
<td>glyph</td>
<td>20</td>
</tr>
<tr>
<td>gn</td>
<td>364</td>
</tr>
<tr>
<td>grammar</td>
<td>338</td>
</tr>
<tr>
<td>graphic rendition, setting</td>
<td>359</td>
</tr>
<tr>
<td>graphics, dot-matrix</td>
<td>374</td>
</tr>
<tr>
<td>graphics, line-drawing</td>
<td>361</td>
</tr>
<tr>
<td>half line cursor movement</td>
<td>364</td>
</tr>
<tr>
<td>half-bright character</td>
<td>358</td>
</tr>
<tr>
<td>halfdelay()</td>
<td>113</td>
</tr>
<tr>
<td>has_colors</td>
<td>56</td>
</tr>
<tr>
<td>has_colors()</td>
<td>114</td>
</tr>
<tr>
<td>has_ic()</td>
<td>115</td>
</tr>
<tr>
<td>has_il</td>
<td>115</td>
</tr>
<tr>
<td>Hazeltine</td>
<td>365</td>
</tr>
<tr>
<td>hc</td>
<td>353</td>
</tr>
<tr>
<td>Index</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>hd ..........................................................</td>
<td>364</td>
</tr>
<tr>
<td>header line in terminfo ................................................</td>
<td>337</td>
</tr>
<tr>
<td>headers .........................................................</td>
<td>305</td>
</tr>
<tr>
<td>Heathkit H19 (example) ..................................................</td>
<td>361</td>
</tr>
<tr>
<td>Hewlett-Packard ..........................................................</td>
<td>363</td>
</tr>
<tr>
<td>Hewlett-Packard 2621 .......................................................</td>
<td>363</td>
</tr>
<tr>
<td>.......................... keypad ........................................</td>
<td>360</td>
</tr>
<tr>
<td>.......................... magic cookie glitch ..................</td>
<td>358</td>
</tr>
<tr>
<td>Hewlett-Packard 2645 (example) .........................................</td>
<td>355</td>
</tr>
<tr>
<td>high-order bit, setting ................................................</td>
<td>365</td>
</tr>
<tr>
<td>highlighting .........................................................</td>
<td>358</td>
</tr>
<tr>
<td>hline() ..................................................................</td>
<td>116</td>
</tr>
<tr>
<td>hline_set() ...........................................................</td>
<td>117</td>
</tr>
<tr>
<td>hls ........................................................................</td>
<td>363</td>
</tr>
<tr>
<td>home ......................................................................</td>
<td>355</td>
</tr>
<tr>
<td>hpa ..........................................................................</td>
<td>355</td>
</tr>
<tr>
<td>hs ...........................................................................</td>
<td>361</td>
</tr>
<tr>
<td>ht ...........................................................................</td>
<td>360</td>
</tr>
<tr>
<td>hts ........................................................................</td>
<td>360</td>
</tr>
<tr>
<td>hu ..........................................................................</td>
<td>364</td>
</tr>
<tr>
<td>hz ...........................................................................</td>
<td>365</td>
</tr>
<tr>
<td>ich .........................................................................</td>
<td>357</td>
</tr>
<tr>
<td>ich1 .......................................................................</td>
<td>357</td>
</tr>
<tr>
<td>idcok() ..................................................................</td>
<td>118</td>
</tr>
<tr>
<td>idlok .................................................................</td>
<td>62</td>
</tr>
<tr>
<td>idlok() ..................................................................</td>
<td>119</td>
</tr>
<tr>
<td>if ...........................................................................</td>
<td>360</td>
</tr>
<tr>
<td>il ...........................................................................</td>
<td>356</td>
</tr>
<tr>
<td>ill ..........................................................................</td>
<td>356</td>
</tr>
<tr>
<td>immedok() ............................................................</td>
<td>120</td>
</tr>
<tr>
<td>implementation-defined ................................................</td>
<td>4</td>
</tr>
<tr>
<td>inch() ....................................................................</td>
<td>123</td>
</tr>
<tr>
<td>inchstr .................................................................</td>
<td>124</td>
</tr>
<tr>
<td>inchstr .................................................................</td>
<td>124</td>
</tr>
<tr>
<td>ind .........................................................................</td>
<td>353, 356</td>
</tr>
<tr>
<td>delays....................................................................</td>
<td>361</td>
</tr>
<tr>
<td>independence of print modes assumed ................................</td>
<td>372</td>
</tr>
<tr>
<td>indn .................................................................</td>
<td>353, 356</td>
</tr>
<tr>
<td>infocmp ...............................................................</td>
<td>324</td>
</tr>
<tr>
<td>initc ......................................................................</td>
<td>363</td>
</tr>
<tr>
<td>initialization ..........................................................</td>
<td>27, 360</td>
</tr>
<tr>
<td>initialization string ...................................................</td>
<td>27</td>
</tr>
<tr>
<td>initialize a color-pair ...............................................</td>
<td>363</td>
</tr>
<tr>
<td>initp .....................................................................</td>
<td>363</td>
</tr>
<tr>
<td>initscr() ..................................................................</td>
<td>126</td>
</tr>
<tr>
<td>init_color ............................................................</td>
<td>.56</td>
</tr>
<tr>
<td>init_color() ............................................................</td>
<td>125</td>
</tr>
<tr>
<td>init_pair ...............................................................</td>
<td>56, 125</td>
</tr>
<tr>
<td>innstr() ...................................................................</td>
<td>128</td>
</tr>
<tr>
<td>innwstr() .............................................................</td>
<td>130</td>
</tr>
<tr>
<td>insch() ...................................................................</td>
<td>134</td>
</tr>
</tbody>
</table>
Index

insdelln() ................................................................................................. 135
insert
delay per line ....................................................................................... 350
effect on straddling character .............................................................. 20
resulting rendition .............................................................................. 21
insert/delete character ........................................................................ 357
insert/delete line ................................................................................ 356
insertion ................................................................................................ 19
insertln() .............................................................................................. 136
insnstr() ............................................................................................... 137
insstr................................................................................................. 137
instr................................................................................................. 128
instr() ............................................................................................... 138
ins_nwstr() ....................................................................................... 132
ins_wch() .......................................................................................... 133
ins_wstr ............................................................................................. 132
interfaces
implementation .................................................................................... 9
system ................................................................................................. 305
use ........................................................................................................ 9
internationalization ............................................................................. 2
intrflush() ........................................................................................ 139
invis ...................................................................................................... 358
invisible text ......................................................................................... 358
inwstr ................................................................................................. 130
inwstr() ............................................................................................. 140
in_wch() ........................................................................................... 121
in_wchnstr() ..................................................................................... 122
in_wchstr .......................................................................................... 122
ip ........................................................................................................... 357
iprog ...................................................................................................... 360
is1, is2, is3 .......................................................................................... 360
isendwin() ......................................................................................... 143
ISO/IEC 6429: 1992 ............................................................................. 338
is_linetouched() ................................................................................. 141
is_wintouched .................................................................................... 141
it .......................................................................................................... 360
italic ..................................................................................................... 372
ka1, ka3 ............................................................................................... 360
kb2 ....................................................................................................... 360
kbs ......................................................................................................... 360
kc1, kc3 ............................................................................................... 360
kclr ........................................................................................................ 360
kctab ................................................................................................. 360
kcub1 ................................................................................................. 360
kcud1 ................................................................................................. 360
kcuf1 ................................................................................................. 360
kcuu1 ................................................................................................. 360
kdc1 ................................................................................................. 360
kd1l ................................................................................................. 360
ked ........................................................................................................ 360
kel ........................................................................................................ 360
Index

keyname() ................................................................................................................................. 144
keypad ........................................................................................................................................ 360
keypad() .................................................................................................................................. 145
key_prefix .................................................................................................................................. 338
key_name ................................................................................................................................... 144
kfo, kf1, and so on .................................................................................................................. 360
khome ....................................................................................................................................... 360
khts ............................................................................................................................................. 360
kich1 .......................................................................................................................................... 360
kill ............................................................................................................................................... 360
kill_character ............................................................................................................................ 379
killchar ....................................................................................................................................... 89
killchar() .................................................................................................................................... 146
killwchar ........................................................................................................................................ 89, 146
kind ............................................................................................................................................. 360
kl .................................................................................................................................................. 360
km ................................................................................................................................................ 365
knp ................................................................................................................................................. 360
kpp ............................................................................................................................................... 360
kri .................................................................................................................................................. 360
krmir .......................................................................................................................................... 360
ktbc .......................................................................................................................................... 360
last entry in terminfo ............................................................................................................... 349
LC_CTYPE .................................................................................................................................. 15
Lear Siegler ADM-3 (example) ............................................................................................ 354
leaveok ....................................................................................................................................... 62
leaveok() ..................................................................................................................................... 147
leaveok() ....................................................................................................................................... 148
left and top edge ...................................................................................................................... 353
left margin .................................................................................................................................. 371
left-to-right writing .................................................................................................................. 3
legacy ............................................................................................................................................. 4
length of line, effect on print rate ............................................................................................ 376
letter-quality ............................................................................................................................ 376
lexical conventions .................................................................................................................... 339
If ................................................................................................................................................... 353
lf0, lf1, and so on .................................................................................................................... 360
lh.................................................................................................................................................. 360
line drawing character .............................................................................................................. 14
line feed ....................................................................................................................................... 371
line graphics .............................................................................................................................. 361
line-drawing macros .................................................................................................................. 308
line/column coordinate ........................................................................................................... 18
LINES ......................................................................................................................................... 32
lines ............................................................................................................................................. 353
LINES .......................................................................................................................................... 377
lines on screen ........................................................................................................................ 353
Il ..................................................................................................................................................... 353
lm .................................................................................................................................................. 365
locale ......................................................................................................................................... 1
locale-specific ........................................................................................................................... 15
long name of device .................................................................................................................. 337
# Index

<table>
<thead>
<tr>
<th>Function/Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>longname()</td>
<td>149</td>
</tr>
<tr>
<td>lpi</td>
<td>375</td>
</tr>
<tr>
<td>lpi[x]</td>
<td>368</td>
</tr>
<tr>
<td>LSI ADM-3a (example)</td>
<td>355</td>
</tr>
<tr>
<td>lw</td>
<td>360</td>
</tr>
<tr>
<td>macros</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>line-drawing</td>
<td>308</td>
</tr>
<tr>
<td>maddr</td>
<td>369</td>
</tr>
<tr>
<td>magic cookie glitch</td>
<td>358</td>
</tr>
<tr>
<td>mandatory delay</td>
<td>350</td>
</tr>
<tr>
<td>manipulation of window</td>
<td>14</td>
</tr>
<tr>
<td>margin</td>
<td>353, 371</td>
</tr>
<tr>
<td>may</td>
<td>4</td>
</tr>
<tr>
<td>mc0, mc4, and so on</td>
<td>365</td>
</tr>
<tr>
<td>mcs</td>
<td>367-368</td>
</tr>
<tr>
<td>mcub[1]</td>
<td>369</td>
</tr>
<tr>
<td>mcud[1]</td>
<td>369</td>
</tr>
<tr>
<td>mcuf[1]</td>
<td>369</td>
</tr>
<tr>
<td>mcuu[1]</td>
<td>369</td>
</tr>
<tr>
<td>mcu[b/d/f/u][1]</td>
<td>370</td>
</tr>
<tr>
<td>media copy string</td>
<td>365</td>
</tr>
<tr>
<td>meta key</td>
<td>365</td>
</tr>
<tr>
<td>meta()</td>
<td>150</td>
</tr>
<tr>
<td>mgc</td>
<td>361, 372</td>
</tr>
<tr>
<td>mhpa</td>
<td>369</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Motion, automatic</td>
<td>372</td>
</tr>
<tr>
<td>move()</td>
<td>151</td>
</tr>
<tr>
<td>mrcup</td>
<td>354</td>
</tr>
<tr>
<td>msgr</td>
<td>358</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi-byte character</td>
<td></td>
</tr>
<tr>
<td>function naming</td>
<td>25</td>
</tr>
<tr>
<td>multi-column character</td>
<td>16</td>
</tr>
<tr>
<td>multiple character functions</td>
<td>25</td>
</tr>
<tr>
<td>must</td>
<td>4</td>
</tr>
<tr>
<td>mv</td>
<td>152</td>
</tr>
<tr>
<td>mv prefix</td>
<td>25</td>
</tr>
<tr>
<td>position arguments</td>
<td>18</td>
</tr>
<tr>
<td>mvaddch</td>
<td>36</td>
</tr>
<tr>
<td>mvaddch()</td>
<td>156</td>
</tr>
<tr>
<td>mvaddchnstr</td>
<td>37, 157</td>
</tr>
<tr>
<td>mvaddchstr</td>
<td>37</td>
</tr>
<tr>
<td>mvaddchstr()</td>
<td>157</td>
</tr>
</tbody>
</table>
### Index

- `mvaddinustr` ...............................................................................................................................................
  
- `mvaddinustr()` ...........................................................................................................................................
  
- `mvaddnwstr` ................................................................................................................................................
  
- `mvaddnwstr()` ............................................................................................................................................
  
- `mvaddstr` .......................................................................................................................................................
  
- `mvaddwstr` .......................................................................................................................................................
  
- `mvadd_wch` ....................................................................................................................................................
  
- `mvadd_wch()` ..............................................................................................................................................
  
- `mvadd_wchstr` ...............................................................................................................................................54
  
- `mvadd_wchstr()` .........................................................................................................................................54
  
- `mvchgat` .......................................................................................................................................................60
  
- `mvchgat()` ....................................................................................................................................................60
  
- `mvcur()` ......................................................................................................................................................160
  
- `mvdelch` .....................................................................................................................................................161
  
- `mvdelch()` ....................................................................................................................................................76
  
- `mvderwin()` ................................................................................................................................................94
  
- `mvgetch` .....................................................................................................................................................102
  
- `mvgetch()` ...................................................................................................................................................165
  
- `mvgetnstr` ..................................................................................................................................................107
  
- `mvgetnstr()` ..............................................................................................................................................167
  
- `mvgetn_wstr` ...............................................................................................................................................105
  
- `mvgetn_wstr()` ..........................................................................................................................................166
  
- `mvgetstr` ....................................................................................................................................................107
  
- `mvget_wch` ..................................................................................................................................................94
  
- `mvget_wch()` ...........................................................................................................................................164
  
- `mvget_wstr` .............................................................................................................................................105
  
- `mvhline` .....................................................................................................................................................116
  
- `mvhline()` ..................................................................................................................................................168
  
- `mvhline_set` ...............................................................................................................................................117
  
- `mvhline_set()` ..........................................................................................................................................169
  
- `mvinch` ......................................................................................................................................................123
  
- `mvinch()` ...................................................................................................................................................172
  
- `mvinchstr` ..................................................................................................................................................124
  
- `mvinchstr()` .............................................................................................................................................173
  
- `mvinnstr` ..................................................................................................................................................128
  
- `mvinnstr()` .............................................................................................................................................174
  
- `mvinnwstr` ..................................................................................................................................................130
  
- `mvinnwstr()` .............................................................................................................................................175
  
- `mvinsch` ...................................................................................................................................................134
  
- `mvinsch()` ..................................................................................................................................................178
  
- `mvinsnstr` ..................................................................................................................................................137
  
- `mvinsnstr()` .............................................................................................................................................179
  
- `mvinsstr` ..................................................................................................................................................137
  
- `mvinsstr()` .............................................................................................................................................179
  
- `mvins` .......................................................................................................................................................128
  
- `mvins()` ....................................................................................................................................................174
  
- `mvins_nwstr` .............................................................................................................................................132
  
- `mvins_nwstr()` .........................................................................................................................................176
  
- `mvins_wch` ..................................................................................................................................................133
  
- `mvins_wch()` ..........................................................................................................................................177
  
- `mvins_wstr` .............................................................................................................................................132
  
- `mvinswstr` ................................................................................................................................................130
  
- `mvinswstr()` ..........................................................................................................................................175
Index

mvwvline_set .............................................................................................................. 117, 169
n infix ....................................................................................................................... 25
name of capability ................................................................................................. 338
name of device ...................................................................................................... 377
name space
   X/Open .................................................................................................................. 10
naming ..................................................................................................................... 24
napms() ................................................................................................................. 183
ncv ....................................................................................................................... 363-364
near-letter-quality .............................................................................................. 376
nel ........................................................................................................................ 353
network terminal ................................................................................................. 364
networked asynchronous terminal ................................................................... 27
newline ................................................................................................................... 353
   special processing ............................................................................................ 21
newpad() ............................................................................................................... 184
newterm ............................................................................................................... 126
newterm() .......................................................................................................... 186
newwin .................................................................................................................. 80
newwin() .............................................................................................................. 187
nl() .................................................................................................................... 188
nlab ..................................................................................................................... 360
no ....................................................................................................................... 189
nocbreak ............................................................................................................ 59
nocbreak() .......................................................................................................... 190
nodelay() .......................................................................................................... 191
noecho .................................................................................................................. 84
noecho() ............................................................................................................. 192
non-spacing character ....................................................................................... 16
non-spacing characters ...................................................................................... 3
non-standard terminal ....................................................................................... 27
nonl ....................................................................................................................... 188
nonl() .................................................................................................................. 193
noqiflush() ......................................................................................................... 194
noraw .................................................................................................................. 59, 190
notimeout mode ............................................................................................... 22
notimeout() ...................................................................................................... 195
npc ..................................................................................................................... 364
npins .................................................................................................................... 374
nnrnc ................................................................................................................... 356
null chtype ......................................................................................................... 379
null wide-character code .................................................................................. 379
numeric capability ........................................................................................... 338
OB ....................................................................................................................... 5
oc ...................................................................................................................... 363
octal specification in terminfo ......................................................................... 350
op ....................................................................................................................... 363
optimization ..................................................................................................... 1
orc ....................................................................................................................... 367
   implied change to ......................................................................................... 368
orhi ..................................................................................................................... 367
   implied change to ......................................................................................... 368
Index

origin.......................................................................................................................................................18
orl............................................................................................................................................................367
implied change to.................................................................................................................................368
orphaned character ...............................................................................................................................18
orphaned column .................................................................................................................................18
orv ............................................................................................................................................................367
implied change to.................................................................................................................................368
os.............................................................................................................................................................353, 358
overlapping ..............................................................................................................................................20
overlay().................................................................................................................................................
overstrike..................................................................................................................................................353
overwrite..................................................................................................................................................196
overwriting..............................................................................................................................................18, 20
prefix.........................................................................................................................................................24-25
pad...............................................................................................................................................................
functions that use..................................................................................................................................25
pad character ..........................................................................................................................................364
padding.......................................................................................................................................................
padding character ....................................................................................................................................350
page eject ..................................................................................................................................................364
pairs...........................................................................................................................................................363
pair_content ..........................................................................................................................................354
pair_content().......................................................................................................................................197
PAIR_NUMBER .....................................................................................................................................197
parametrized string ................................................................................................................................354
parent window ......................................................................................................................................14, 379
patch.........................................................................................................................................................
PC terminal emulator .........................................................................................................................363
pechochar()............................................................................................................................................
pecho wchar ..........................................................................................................................................198
period in terminfo ................................................................................................................................351
Perkin-Elmer Owl (example)...............................................................................................................357
pfkey.........................................................................................................................................................360
pfloc..........................................................................................................................................................360
pfx...............................................................................................................................................................
pln...............................................................................................................................................................360
pnoutrefresh.........................................................................................................................................184
pnoutrefresh().......................................................................................................................................199
pop-up window .......................................................................................................................................
20
porder.......................................................................................................................................................374
position
  current or specified .................................................................................................................................25
postfix.......................................................................................................................................................354
prefix on function/argument .........................................................................................................................24
prefresh.......................................................................................................................................................
print quality ...............................................................................................................................................376
printer resolution .....................................................................................................................................366
printer specification in terminfo ..............................................................................................................366
printing rate ..............................................................................................................................................376
printw ........................................................................................................................................................180
printw()..................................................................................................................................................200

X/Open Curses, Issue 7 395
## Index

<table>
<thead>
<tr>
<th>Property</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>18</td>
</tr>
<tr>
<td>rendition</td>
<td>18</td>
</tr>
<tr>
<td>window</td>
<td>17</td>
</tr>
<tr>
<td>proportional delay</td>
<td>350</td>
</tr>
<tr>
<td>proportional printing</td>
<td>366</td>
</tr>
<tr>
<td>prot</td>
<td>358</td>
</tr>
<tr>
<td>protected text</td>
<td>358</td>
</tr>
<tr>
<td>protocol (xon/xoff)</td>
<td>353</td>
</tr>
<tr>
<td>putp()</td>
<td>201</td>
</tr>
<tr>
<td>putwin()</td>
<td>111</td>
</tr>
<tr>
<td>putwin()</td>
<td>202</td>
</tr>
<tr>
<td>qiflush()</td>
<td>194</td>
</tr>
<tr>
<td>qiflush()</td>
<td>203</td>
</tr>
<tr>
<td>quality of printing</td>
<td>376</td>
</tr>
<tr>
<td>raster graphics</td>
<td>374</td>
</tr>
<tr>
<td>raw</td>
<td>59</td>
</tr>
<tr>
<td>raw()</td>
<td>204</td>
</tr>
<tr>
<td>rbim</td>
<td>374</td>
</tr>
<tr>
<td>rc</td>
<td>356, 361</td>
</tr>
<tr>
<td>inclusion in tsl/fsl</td>
<td>361</td>
</tr>
<tr>
<td>rcsd</td>
<td>373</td>
</tr>
<tr>
<td>reading subwindow</td>
<td></td>
</tr>
<tr>
<td>effect on straddling character</td>
<td>20</td>
</tr>
<tr>
<td>redrawwin()</td>
<td>205</td>
</tr>
<tr>
<td>reference pages</td>
<td>29</td>
</tr>
<tr>
<td>format</td>
<td>6</td>
</tr>
<tr>
<td>refresh</td>
<td>82</td>
</tr>
<tr>
<td>clears touched flag</td>
<td>14</td>
</tr>
<tr>
<td>refresh()</td>
<td>206</td>
</tr>
<tr>
<td>relocation of cursor</td>
<td>19</td>
</tr>
<tr>
<td>rendition</td>
<td>16, 359, 379</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>21</td>
</tr>
<tr>
<td>rep</td>
<td>364</td>
</tr>
<tr>
<td>replacing characters</td>
<td>18</td>
</tr>
<tr>
<td>resety()</td>
<td>208</td>
</tr>
<tr>
<td>reset_prog_mode</td>
<td>72</td>
</tr>
<tr>
<td>reset_prog_mode()</td>
<td>207</td>
</tr>
<tr>
<td>reset_shell_mode</td>
<td>72, 207</td>
</tr>
<tr>
<td>resolution</td>
<td>366</td>
</tr>
<tr>
<td>resolution, effect of changing</td>
<td>375</td>
</tr>
<tr>
<td>restarterm</td>
<td>73</td>
</tr>
<tr>
<td>restarterm()</td>
<td>209</td>
</tr>
<tr>
<td>restoring subwindow</td>
<td>20</td>
</tr>
<tr>
<td>rev</td>
<td>358</td>
</tr>
<tr>
<td>reverse Polish</td>
<td>354</td>
</tr>
<tr>
<td>reverse-video screen</td>
<td>358</td>
</tr>
<tr>
<td>rf</td>
<td>360</td>
</tr>
<tr>
<td>rfi</td>
<td>364</td>
</tr>
<tr>
<td>ri</td>
<td>353, 356</td>
</tr>
</tbody>
</table>
Index

right margin ..............................................................................................................................................371
right-to-left writing ...............................................................................................................3
rin .................................................................................................................................353, 356
ripoffline() .......................................................................................................................................210
ritm .................................................................372
rim .................................................................370
rmacs ..........................................................358
rmcup ..........................................................356
rmdc ..............................................................357
rmicm ..........................................................370
rmir ..............................................................357
rmkx ..............................................................360
rmln ..............................................................360
rm .................................................................365
rmp .................................................................357
rmso ..............................................................358
rmul ..............................................................358
rmxon ...........................................................365
rounding .........................................................................................................................366
row or column cursor addressing .........................................................................................355
RPN ...........................................................................................................................354
rs1, rs2 ..........................................................360
rshm ..............................................................372
rsubm .............................................................372
rsupm .............................................................372
rum ..............................................................370
rwdm ..............................................................372
sam .................................................................370
savetty ...............................................................208
savetty() ..............................................................211
sbim .................................................................374
sc ..........................................................................................................................356, 361
 inclusion in tsl/fsl ...........................................................................................................361
scanw ..............................................................................................................................181
scanw() .................................................................212
scp ...............................................................................................................................363
tscreen ...............................................................14, 353
SCREEN .................................................................379
tscreen blink ...................................................................................................................358
tscroll() ...........................................................................................................................215
tscroll .........................................................................................................................................215
effect on straddling character ....................................................................................................20
tscrolling .....................................................................................................................................353
ntscrolling region ...................................................................................................................356
ntscrolllok ..................................................................................................................................62
ntscrolllok() ..........................................................................................................................216
tscr_dump() ..........................................................................................................................213
tscr_init ...................................................................................................................................213
tscr_restore ...........................................................................................................................213
tscr_set ......................................................................................................................................213
ntscs ...........................................................................................................................................373
ntscsd .........................................................................................................................................373
sdreq ....................................................................................................................................................376
search path for TERM .....................................................................................................................377
setab ................................................................................................................................................363
setaf ................................................................................................................................................363
setb ................................................................................................................................................363
setchar() .....................................................................................................................................219
setf ................................................................................................................................................363
setscreen ....................................................................................................................................62
setscrreg() ................................................................................................................................220
settable scrolling region .............................................................................................................356
setupterm ....................................................................................................................................73
setupterm() ................................................................................................................................221
set_curterm ..................................................................................................................................73
set_curterm() .............................................................................................................................217
set_term() ...................................................................................................................................218
sg .....................................................................................................................................................359
sg0 ..................................................................................................................................................358
shaded text ...................................................................................................................................5
shadow .........................................................................................................................................372
shadowing .....................................................................................................................................372
shall ..............................................................................................................................................4
sharing definition in terminfo ...................................................................................................366
should ......................................................................................................................................366
signals .........................................................................................................................................13-14
similar terminal ...........................................................................................................................366
single-byte character
  function naming .......................................................................................................................25
sitm .............................................................................................................................................372
slash
  in terminfo .............................................................................................................................350
slk_attroff() ................................................................................................................................222
slk_attron ...................................................................................................................................222
slk_attrset ..................................................................................................................................222
slk_attr_off ................................................................................................................................222
slk_attr_on ..................................................................................................................................222
slk_attr_set ................................................................................................................................222
slk_clear .....................................................................................................................................222
slk_color .......................................................................................................................................222
slk_init .........................................................................................................................................222
slk_label ......................................................................................................................................222
slk_noutrefresh ..........................................................................................................................222
slk_refresh ................................................................................................................................222
slk_restore ...................................................................................................................................222
slk_set ..........................................................................................................................................222
slk_touch .....................................................................................................................................222
slk_wset .....................................................................................................................................222
slm ................................................................................................................................................370
sm ..................................................................................................................................................370
smacs ..........................................................................................................................................356
smc ...............................................................................................................................................357
smg .............................................................................................................................................371
Index

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>smicm</td>
<td>370</td>
</tr>
<tr>
<td>smir</td>
<td>357</td>
</tr>
<tr>
<td>smkx</td>
<td>360</td>
</tr>
<tr>
<td>smln</td>
<td>360</td>
</tr>
<tr>
<td>smm</td>
<td>365</td>
</tr>
<tr>
<td>smso</td>
<td>358</td>
</tr>
<tr>
<td>smul</td>
<td>358</td>
</tr>
<tr>
<td>smxon</td>
<td>365</td>
</tr>
<tr>
<td>snlq</td>
<td>376</td>
</tr>
<tr>
<td>snrmq</td>
<td>376</td>
</tr>
<tr>
<td>space</td>
<td>337</td>
</tr>
<tr>
<td>space character</td>
<td></td>
</tr>
<tr>
<td>resulting rendition</td>
<td>21</td>
</tr>
<tr>
<td>spacing complex character</td>
<td>17</td>
</tr>
<tr>
<td>spacing of characters</td>
<td>366</td>
</tr>
<tr>
<td>special characters</td>
<td>20</td>
</tr>
<tr>
<td>special keys</td>
<td>360</td>
</tr>
<tr>
<td>special mode</td>
<td>356</td>
</tr>
<tr>
<td>special mode of device</td>
<td>377</td>
</tr>
<tr>
<td>speed of printing</td>
<td>376</td>
</tr>
<tr>
<td>spinh</td>
<td>374</td>
</tr>
<tr>
<td>spinv</td>
<td>374</td>
</tr>
<tr>
<td>sshm</td>
<td>372</td>
</tr>
<tr>
<td>ssbm</td>
<td>372</td>
</tr>
<tr>
<td>ssupm</td>
<td>372</td>
</tr>
<tr>
<td>stack in terminfo</td>
<td>354</td>
</tr>
<tr>
<td>standend()</td>
<td>224</td>
</tr>
<tr>
<td>standout</td>
<td>224</td>
</tr>
<tr>
<td>standout mode</td>
<td>358</td>
</tr>
<tr>
<td>start_color</td>
<td>56</td>
</tr>
<tr>
<td>start_color()</td>
<td>225</td>
</tr>
<tr>
<td>status line</td>
<td>361</td>
</tr>
<tr>
<td>stdscr</td>
<td>14, 226</td>
</tr>
<tr>
<td>straddling character</td>
<td>20</td>
</tr>
<tr>
<td>string capability</td>
<td>338</td>
</tr>
<tr>
<td>string, parametrized</td>
<td>354</td>
</tr>
<tr>
<td>subcs</td>
<td>372</td>
</tr>
<tr>
<td>subpad</td>
<td>15, 184</td>
</tr>
<tr>
<td>subpad()</td>
<td>227</td>
</tr>
<tr>
<td>subscript</td>
<td>372</td>
</tr>
<tr>
<td>characters available</td>
<td>372</td>
</tr>
<tr>
<td>subwin</td>
<td>80</td>
</tr>
<tr>
<td>subwin()</td>
<td>228</td>
</tr>
<tr>
<td>overview</td>
<td>14</td>
</tr>
<tr>
<td>subwindow</td>
<td>14, 379</td>
</tr>
<tr>
<td>character straddling border</td>
<td>20</td>
</tr>
<tr>
<td>sum</td>
<td>370</td>
</tr>
<tr>
<td>supcs</td>
<td>372</td>
</tr>
<tr>
<td>superscript</td>
<td>372</td>
</tr>
<tr>
<td>characters available</td>
<td>372</td>
</tr>
<tr>
<td>swidm</td>
<td>372</td>
</tr>
</tbody>
</table>
switch ......................................................................................................................................................... 364
synchronous terminal ................................................................................................................................... 27
synck() .......................................................................................................................................................... 229
system interfaces ......................................................................................................................................... 305
tab .............................................................................................................................................................. 360
delays ........................................................................................................................................................... 361
expansion ................................................................................................................................................... 355
special processing ....................................................................................................................................... 21
use in terminfo .......................................................................................................................................... 337
tab stop ....................................................................................................................................................... 27
tbc ............................................................................................................................................................. 360
Tektronix
model of color specification ....................................................................................................................... 363
Tektronix 4025
command character ...................................................................................................................................... 364
Tektronix 4025 (example) ........................................................................................................................... 355
Teleray
destructive tab ................................................................................................................................ ........ 365
Teleray 1061 (example) ................................................................................................................................ 358
termattr() ................................................................................................................................................... 230
terminal ....................................................................................................................................................... 15
terminal emulator ....................................................................................................................................... 363
terminal-independence .............................................................................................................................. 1, 337
terminfo ...................................................................................................................................................... 337
TERMINFO .................................................................................................................................................. 377
terminfo
format .......................................................................................................................................................... 337
terminology .................................................................................................................................................. 4
termmame() ................................................................................................................................................. 231
thread-safety ............................................................................................................................................... 13
throughput .................................................................................................................................................. 376
tic ................................................................................................................................................................ 328
tigetflag() .................................................................................................................................................. 232
tigetnum ..................................................................................................................................................... 232
tigetstr .......................................................................................................................................................... 232
tilde, inability to display .............................................................................................................................. 365
timeout .......................................................................................................................................................... 195
timeout() ..................................................................................................................................................... 235
tiparm ............................................................................................................................................................ 232
tiparm() ....................................................................................................................................................... 236
top and left edge ......................................................................................................................................... 353
touch ........................................................................................................................................................... 379
touched ......................................................................................................................................................... 14
touchline ..................................................................................................................................................... 141
touchline() .................................................................................................................................................. 237
touchwin .................................................................................................................................................... 141, 237
tparm ............................................................................................................................................................ 232
tparm() ........................................................................................................................................................ 238
tput ............................................................................................................................................................... 330
tputs() .......................................................................................................................................................... 239
truncation ..................................................................................................................................................... 19
tst ................................................................................................................................................................ 361
Index

TVI 912 (example) ................................................................. 358
typeahead() ........................................................................... 240
uc ....................................................................................... 358
ul ....................................................................................... 358
unctrl() .............................................................................. 241
undefined ............................................................................. 5
underline cursor ............................................................... 358
underlining ......................................................................... 358
ungetch() ............................................................................ 242
unget_wch .......................................................................... 242
uniqueness of terminfo aliases .......................................... 337
unspecified ........................................................................... 5
untic .................................................................................... 355
untouchwin ......................................................................... 141
untouchwin() .................................................................... 243
update
  sets touched flag ............................................................. 14
use ..................................................................................... 366
user preference for use of device ...................................... 377
use_env() .......................................................................... 244
utilities ............................................................................... 323
variability in print rate ..................................................... 376
variable-width font .......................................................... 366
vertical bar
  use in terminfo ................................................................. 337
vi
  use of terminfo ................................................................. 337
vidattr() ............................................................................. 245
video attribute ................................................................. 16
video enhancement, turn off ........................................... 358
vidputs ............................................................................... 245
vid_attr ............................................................................ 245
vid_puts ............................................................................ 245
virtual terminal ............................................................... 364
visible bell .......................................................................... 358
vline .................................................................................. 116
vline() .............................................................................. 247
vline_set ........................................................................... 117
vline_set() ........................................................................ 248
vpa .................................................................................... 355
vt ...................................................................................... 364
VT100
  delayed line wrap ............................................................ 365
  line drawing .................................................................... 361
  scrolling region ............................................................. 356
  status line ....................................................................... 361
vw_printw() ....................................................................... 249
vw_scanw() ....................................................................... 250
w ....................................................................................... 251
w infix ................................................................................ 25
w prefix .............................................................................. 24
waddch .............................................................................. 36
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>waddch()</td>
<td>255</td>
</tr>
<tr>
<td>waddchnstr</td>
<td>37, 256</td>
</tr>
<tr>
<td>waddcstr</td>
<td>37</td>
</tr>
<tr>
<td>waddcstr()</td>
<td>256</td>
</tr>
<tr>
<td>waddnstr</td>
<td>38</td>
</tr>
<tr>
<td>waddnstr()</td>
<td>257</td>
</tr>
<tr>
<td>waddnwstr</td>
<td>40</td>
</tr>
<tr>
<td>waddnwstr()</td>
<td>258</td>
</tr>
<tr>
<td>waddstr</td>
<td>38, 257</td>
</tr>
<tr>
<td>waddwstr</td>
<td>40, 258</td>
</tr>
<tr>
<td>wadd_wch</td>
<td>33</td>
</tr>
<tr>
<td>wadd_wch()</td>
<td>253</td>
</tr>
<tr>
<td>wadd_wchstr</td>
<td>34</td>
</tr>
<tr>
<td>wadd_wchstr()</td>
<td>254</td>
</tr>
<tr>
<td>wadd_wchstr()</td>
<td>34, 254</td>
</tr>
<tr>
<td>wattr_off</td>
<td>41, 259</td>
</tr>
<tr>
<td>wattr_on</td>
<td>41, 259</td>
</tr>
<tr>
<td>wattr_set</td>
<td>41, 259</td>
</tr>
<tr>
<td>wattr_set</td>
<td>41, 259</td>
</tr>
<tr>
<td>wbkgd</td>
<td>46</td>
</tr>
<tr>
<td>wbkgd()</td>
<td>261</td>
</tr>
<tr>
<td>wbkgdset</td>
<td>46, 261</td>
</tr>
<tr>
<td>wbkgnd</td>
<td>48</td>
</tr>
<tr>
<td>wbkgnd()</td>
<td>262</td>
</tr>
<tr>
<td>wbkgndset</td>
<td>48, 262</td>
</tr>
<tr>
<td>wborder</td>
<td>50</td>
</tr>
<tr>
<td>wborder()</td>
<td>263</td>
</tr>
<tr>
<td>wborder_set</td>
<td>52</td>
</tr>
<tr>
<td>wborder_set()</td>
<td>264</td>
</tr>
<tr>
<td>wchgt</td>
<td>60</td>
</tr>
<tr>
<td>wchgt()</td>
<td>265</td>
</tr>
<tr>
<td>wclear</td>
<td>61</td>
</tr>
<tr>
<td>wclear()</td>
<td>266</td>
</tr>
<tr>
<td>wclrtobot</td>
<td>64</td>
</tr>
<tr>
<td>wclrtobot()</td>
<td>267</td>
</tr>
<tr>
<td>wclrtoeol</td>
<td>65</td>
</tr>
<tr>
<td>wclrtoeol()</td>
<td>268</td>
</tr>
<tr>
<td>wcolor_set</td>
<td>269</td>
</tr>
<tr>
<td>wcolor_set()</td>
<td>41</td>
</tr>
<tr>
<td>wcredsyncup</td>
<td>229</td>
</tr>
<tr>
<td>wcredsyncup()</td>
<td>270</td>
</tr>
<tr>
<td>wdelc</td>
<td>76</td>
</tr>
<tr>
<td>wdelc()</td>
<td>271</td>
</tr>
<tr>
<td>wdeleteln</td>
<td>77</td>
</tr>
<tr>
<td>wdeleteln()</td>
<td>272</td>
</tr>
<tr>
<td>wechochar</td>
<td>86</td>
</tr>
<tr>
<td>wechochar()</td>
<td>274</td>
</tr>
</tbody>
</table>
Index

wecho_wchar.................................................................................................................................85
wecho_wchar()...................................................................................................................................273
werase.....................................................................................................................................................61, 88, 266
wgetbgnd................................................................................................................................................48, 262
wgetch........................................................................................................................................................102
wgetch()....................................................................................................................................................276
wgetnstr...................................................................................................................................................107
wgetnstr()...............................................................................................................................................278
wgetn_wstr..............................................................................................................................................105
wgetn_wstr()...........................................................................................................................................277
wgetstr......................................................................................................................................................107, 278
wget_wch...................................................................................................................................................94
wget_wch()...............................................................................................................................................275
wget_wstr...................................................................................................................................................105, 277
whcline_set()..........................................................................................................................................280
whline........................................................................................................................................................116
whline().....................................................................................................................................................279
whline_set................................................................................................................................................117
widcs.......................................................................................................................................................368, 372
wide character.........................................................................................................................................372
wide characters.........................................................................................................................................3
wide mode..................................................................................................................................................377
wide-character code (C language)...........................................................................................................379
wide-character string................................................................................................................................379
width of character, variable......................................................................................................................366
will.............................................................................................................................................................5
winch.........................................................................................................................................................123
winch().......................................................................................................................................................283
winchnstr....................................................................................................................................................124
winchnstr()..............................................................................................................................................284
winchstr......................................................................................................................................................124, 284
wind............................................................................................................................................................356
window......................................................................................................................................................14, 356, 380
clipping......................................................................................................................................................14
current or specified..................................................................................................................................24
parent..........................................................................................................................................................14
touched flag...............................................................................................................................................14
window background..................................................................................................................................18
window hierarchy......................................................................................................................................380
window property......................................................................................................................................17
window rendition.......................................................................................................................................18
winnstr........................................................................................................................................................128
winnstr()....................................................................................................................................................285
winnwstr....................................................................................................................................................130
winnwstr()...............................................................................................................................................286
winsch.......................................................................................................................................................134
winsch()....................................................................................................................................................289
winsdelln....................................................................................................................................................135
winsdelln()...............................................................................................................................................290
winseln.......................................................................................................................................................136
winseln()...................................................................................................................................................291
winsnstr......................................................................................................................................................137

X/Open Curses, Issue 7 403
Index

winsnstr() ........................................................................................................................................... 292
winsstr .............................................................................................................................................. 137, 292
winst .................................................................................................................................................. 128, 285
wins_nwstr ....................................................................................................................................... 132
wins_nwstr() ................................................................................................................................... 287
winst_nch ........................................................................................................................................... 133
winst_wch() ....................................................................................................................................... 288
winst_wstr .......................................................................................................................................... 132, 287
winwstr .............................................................................................................................................. 130, 286
win_wch .............................................................................................................................................. 121
win_wch() ........................................................................................................................................... 281
win_wchnstr ..................................................................................................................................... 122
win_wchnstr() .................................................................................................................................. 282
win_wchstr ........................................................................................................................................ 122, 282
withdraw .......................................................................................................................................... 2
wmove ............................................................................................................................................... 151
wmove() ............................................................................................................................................ 293
wnoutrefresh ................................................................................................................................... 82
wnoutrefresh() ................................................................................................................................ 294
wprintw ............................................................................................................................................. 180
wprintw() .......................................................................................................................................... 295
wrap to next line ............................................................................................................................... 353
wrapping .......................................................................................................................................... 19
wredrawln ......................................................................................................................................... 205
wredrawln() ..................................................................................................................................... 296
wrefresh ........................................................................................................................................... 82, 294
wscrl ............................................................................................................................................... 215
wscrl() ............................................................................................................................................ 298
wsescrreg ......................................................................................................................................... 62
wsescrreg() .................................................................................................................................... 299
wsl .................................................................................................................................................... 361
wstandend() ................................................................................................................................... 224
wstandout ........................................................................................................................................ 224, 300
wsyncdown ..................................................................................................................................... 229
wsyncdown() .................................................................................................................................. 301
wsyncup ........................................................................................................................................... 229, 301
wtimeout ......................................................................................................................................... 195
wtimeout() ...................................................................................................................................... 302
wtouchln .......................................................................................................................................... 141
wtouchln() ...................................................................................................................................... 303
wunctrl() ......................................................................................................................................... 304
wvline ............................................................................................................................................. 116, 279
wvline_set ....................................................................................................................................... 117, 280
X/Open name space .......................................................................................................................... 10
xenl ................................................................................................................................................... 365
xhp ................................................................................................................................................... 365
xhpa ................................................................................................................................................... 369
xmc ................................................................................................................................................... 358
xoffc ................................................................................................................................................. 365
Index

xon.....................................................................................................................................................353, 365
  and padding characters......................................................................................................................350
xonc ...........................................................................................................................................................365
xsb..............................................................................................................................................................366
xt.................................................................................................................................................................365
xvpa...........................................................................................................................................................369
y, x pair........................................................................................................................................................
  18
zero-based row/column numbering....................................................................................................354
zero-width character................................................................................................................................
  16
zerom ........................................................................................................................................................371