

# **ncurses**

Elaine de Claro  
Mico Galang  
Fina Mesina  
Ma. Athena Rebong  
Kenneth Talamayan

## ncurses

- The curses package is a subroutine library for terminal-independent screen-painting and input-event handling.

- uses terminfo, which is a database format that can describe the capabilities of thousands of different terminals. uses terminfo format, supports pads, colors, multiple highlights, form characters and function key mapping.

- a freely distributable "clone" of System V Release 4.0 (SVr4) curses

### Advantages of the curses API

- back-portability to character-cell terminals
- simplicity

## History

- The first ancestor of curses was the routines written to provide screen-handling for the game rogue; these used the already-existing termcap database facility for describing terminal capabilities.
- These routines were abstracted into a documented library and first released with the early BSD UNIX versions.
- System III UNIX from Bell Labs featured a rewritten and much-improved curses library. It introduced the terminfo format.
- In the later AT&T System V releases, curses evolved to use more facilities and offer more capabilities, going far beyond BSD curses in power and flexibility.

## History (cont.)

- The package was originated as **pcurses**, written by Pavel Curtis around 1982, maintained by various people through 1986.

- It was later polished (e.g., ANSI prototypes) and re-issued as ncurses 1.8.1 in late 1993 by Zeyd Ben-Halim.

- Subsequent work (through 1.8.8) was driven by Eric Raymond, who eradicated previous signs of authorship with the current copyright notice between 1.8.7 and 1.8.8, early 1995.

- Later, this extended to incorporating the forms and menus libraries written by Juergen Pfeifer, and a panel library written by Warren Tucker.

## CONCEPTS

### *A Window*

- an internal data representation of an image of what a particular rectangular section of the terminal display may look like. The terminal display as a whole could be said to be a window, its dimensions defined by its outermost extremities, those being the side of the cathode ray tube.

- A curses window is not a physical entity. It is only a data representation of how you would like a rectangular portion of the physical screen to look. A window is a preallocated resource stored in the machine's memory. As you manipulate it, nothing actually happens to the physical screen until you are ready to update it.

## CONCEPTS (cont.)

### *A Window (cont.)*

- Curses provides a default window which represents your terminal screen.
- Its size is defined by the dimensions of the terminal screen your curses program is to work with.
- Curses allows you to manipulate several windows individually, or all at the same time.
- It also allows windows to contain windows within themselves, known as sub-windows.
- You can create as many windows as you want;
- the only limitation is the amount of memory available which your program can use.

## CONCEPTS (cont.)

### *The Curses Window*

- The curses internal representation of a window is defined in the data structure:

```
struct _win_st
```

- This structure is defined in the `/usr/include/curses.h` include file and is typedef `WINDOW`.

- It contains all the necessary data and information which curses needs to manage the window on the terminal screen.

- Anything inside or belonging to a window is modifiable.

- Anything outside is undefined and usually illegal.

- It is important to realize that almost all the curses routines totally depend on this window structure.

## CONCEPTS (cont.)

### *The Curses Window (cont.)*

- although window structure is an internal representation of a curses window, it may not necessarily bear any relation to what is really being displayed on the terminal screen
  - The window structure is used solely to hold data and information which describes a window
    - like a buffer area set aside to represent a portion of the screen which you can modify by using the routines provided in the curses library.
    - eg, Hello World Program

## CONCEPTS (cont.)

### *The Terminal Screen*

- When `curses` starts up, the first thing it does is clear the screen it then places the cursor in the home position, which is the top left-hand corner of the screen.
- `Curses` then knows exactly what your physical screen looks like and where the cursor is situated.
- `curses` provides two `WINDOW` data structures, `curscr` and `stdscr`.
- The `curscr` window holds a data representation of what is currently displayed on the real terminal screen,
- and the `stdscr` window is provided as a default window for the programmer to work with.

## CONCEPTS (cont.)

### *The Terminal Screen (cont.)*

- When making the terminal screen look like the `stdscr` window, a `refresh()` call is made to update the `curscr` window
  - it is not good practice to access the `curscr` window directly
  - Changes should be made only to the appropriate window being worked on, and by calling `wrefresh()` on that window, `curses` can be instructed to update the `curscr` window internally
  - The `curscr` window should be treated as a reserved window for the private use of the `curses` routines only.

## CONCEPTS (cont.)

### *The Window Structure Variables*

- The WINDOW structure contains all the necessary information to enable curses to update the terminal screen optimally
  - By keeping state information about a particular window inside its WINDOW data structure, curses can obtain the relevant information it needs to carry out its instructions
  - all the information curses requires to maintain a window is contained within the WINDOW data structure.

## CONCEPTS (cont.)

### *The VARIABLES*

`_cury` and `_curx`

The variables `_cury` and `_curx` contain the current y,x coordinates of the cursor on the window

`_maxy` and `_maxx`

The variables `_maxy` and `_maxx` specify the outermost dimensions of a curses window

`_begy` and `_begx`

The variables `_begy` and `_begx` specify the starting y,x coordinates of the window

## CONCEPTS (cont.)

### *The VARIABLES (cont.)*

`_flags`      The `_flags` variable is used as a bit mask

The following are descriptions of various flags:

`_SUBWIN`      This tells curses that the window is a subwindow.

`_ENDLINE`      This tells curses that the right-most-end of each line in the window is the edge of the screen.

`_FULLWIN`      This tells curses that the window is the size of the physical screen

`_SCROLLWIN`      This tells curses that the terminal will scroll if a character is placed into the lower right edge of the physical screen

## CONCEPTS (cont.)

*The VARIABLES, \_flags (cont.)*

`_FLUSH` This flag is currently unused.

`_ISPAD` This tells curses that the window is a pad;

`_STANDOUT` This tells curses that characters added to the screen are to be displayed in standout mode.

`_NOCHANGE` Used for optimization purposes. When refreshing a window, and if a line on the window has not been changed since the last update curses assumes that there is no need to update it again and so ignores it.

## CONCEPTS (cont.)

*The VARIABLES \_flags (cont.)*

`_WINCHANGED` This tells curses that the window has been modified in some way since it was last updated.

`_WINMOVED` This tells curses that the cursor has been relocated to another position within a window.

All these flags are manipulated internally by the curses routines.

## CONCEPTS (cont.)

*The variables (cont.)*

`_attrs` variable contains various attribute flags relative to a window

`_clear` variable is used within curses to specify whether to clear the terminal screen before the next call to `refresh()`

`_leave` used to tell curses to leave the cursor where it was before the `refresh()` was issued

`_tmarg` and `_bmarg` contain the top and bottom margins of the scrolling region within a window

## CONCEPTS (cont.)

*The variables (cont.)*

<code>scroll()</code>	variable tells curses if the window is allowed to scroll or not
<code>_use_idl</code>	tells curses to use the terminals insert/delete line feature, if the terminal is capable
<code>_use_keypad</code>	tells curses to use the keypad
<code>_use_meta</code>	used to tell curses if it is to operate in <i>meta</i> mode
<code>_nodelay</code>	tells the input routine <code>wgetch()</code> to return without delay if there are no characters waiting in the input queue

## CONCEPTS (cont.)

### *The variables (cont.)*

`_y` this is the two-dimensional array of pointers to lines containing characters that we manipulate in a window

`_firstch` and `_lastch` arrays are to help curses optimize the number of characters on a window it is to update

## CONCEPTS (cont.)

### *The Terminal*

- All interaction between the user and the UNIX operating system is handled by an asynchronous terminal connected through some port
- The normal standard communication method used for these devices is RS-232
- This standard provides a method by which data can be transmitted by the terminal's keyboard and received by the computer
- The only way UNIX knows about a particular device is through a piece of software linked into the kernel called a device driver
- A device driver provides a platform by which a program can read from and write to a device through system entry points provided in the way of C function calls, called system calls.

## CONCEPTS (cont.)

### Structure of the UNIX device interface

- The *tty* device driver essentially controls the data path between the computer and the terminal

### *Terminfo*

The term *terminfo* refers to a group of routines which are built into the curses library and use the capabilities of a terminal. Terminfo also refers to a database of binary files, with each file describing an individual terminal's capabilities.

## *Widgets*

### **CDK**

CDK stands for 'Curses Development Kit' and it currently contains 21 ready to use widgets which facilitate the speedy development of full screen curses programs

## *Widget Types*

### **Alphalist**

Allows a user to select from a list of words, with the ability to narrow the search list by typing in a few characters of the desired word.

### **Buttonbox**

This creates a multiple button widget.

## Widget Types (cont.)

### Calendar

Creates a little simple calendar widget.

### Dialog

Prompts the user with a message, and the user can pick an answer from the buttons provided.

### Entry

Allows the user to enter various types of information.

### File Selector

A file selector built from Cdk base widgets. This example shows how to create more complicated widgets using the Cdk widget library.

## Widget Types (cont.)

### Graph

Draws a graph.

### Histogram

Draws a histogram.

### Item List

Creates a pop up field which allows the user to select one of several choices in a small field. Very useful for things like days of the week or month names.

### Label

Displays messages in a pop up box, or the label can be considered part of the screen.

## Widget Types (cont.)

### Marquee

Displays a message in a scrolling marquee.

### Matrix

Creates a complex matrix with lots of options.

### Menu

Creates a pull-down menu interface.

### Multiple Line Entry

A multiple line entry field. Very useful for long fields. (like a description field)

### Radio List

Creates a radio button list.

## Widget Types (cont.)

### Scale

Creates a numeric scale. Used for allowing a user to pick a numeric value and restrict them to a range of values.

### Scrolling List

Creates a scrolling list/menu list.

### Scrolling Window

Creates a scrolling log file viewer. Can add information into the window while its running. A good widget for displaying the progress of something. (akin to a console window)

## Widget Types (cont.)

### Selection List

Creates a multiple option selection list.

### Slider

Akin to the scale widget, this widget provides a visual slide bar to represent the numeric value.

### Template

Creates a entry field with character sensitive positions. Used for pre-formatted fields like dates and phone numbers.

### Viewer

This is a file/information viewer. Very useful when you need to display loads of information.

## BUILDING AN NCURSES PROGRAM

### *Compiling With the Ncurses Library*

To use ncurses library functions, you have to include ncurses.h and to link the program with ncurses library the flag -lncurses should be added. ncurses.h already includes stdio.h.

## **How to Build an Ncurses Program**

### *Basic Program Structure*

To learn how to create an ncurses program, you need to understand the structure of an ncurses-based program

## *The basic structure of an ncurses program:*

### 1. **command line processing**

Before starting up ncurses, it is a good idea to do any command line processing first. Do not enter into ncurses mode if you are not yet ready to do so.

### 2. **initialize ncurses**

Many internal variables and structures must be set up before ncurses is ready to use. Therefore, ncurses must be initialized. This is done with `initscr ( )` which sets up the provided windows, `curscr` and `stdscr`, initializes the terminal, and then places the program into an in-curses mode.

The basic structure of an ncurses program is as follows:

### 3. **set up desired terminal I/O modes**

If you want to set any special I/O modes in the terminal driver, this should be done after initialization. These modes are normally set up once only, and then remain unmodified throughout program execution.

### 4. **continue processing in ncurses mode**

The program now operating in curses mode can continue processing with the curses facilities. For example, create/delete windows, add characters to a window, get keyboard input, and so on.

The basic structure of an ncurses program is as follows:

5. **end ncurses mode**

When processing is complete and the program is ready to return to the shell (or calling process), the terminal must first be placed back into its original operating modes, the modes that were current before entering into the program. This is done with `endwin ( )`

6. **exit program**

## Compiling with the ncurses Library

To use ncurses library functions, you have to include ncurses.h and to link the program with ncurses library the flag `-lncurses` should be added. ncurses.h already includes `stdio.h`.

**For example:**

To compile: **`gcc -Wall -Werror -o main main.c  
-lncurses`**

To run: **`./main`**

## WINDOW MANAGEMENT

- A Window is an imaginary screen defined by curses system.
- Does not mean a bordered window which you usually see on Win9X platforms.
- When curses is initialized, it creates a default window named `stdscr` which represents your 80x25 (or the size of window in which you are running) screen.

### Creating Windows

If you want to use a window other than the default windows supplied by ncurses (`stdscr` and `curscr`), we need to create it before we can access it. Ncurses provides the function `newwin` (`lines,cols,begy,begx`) for this purpose. The `newwin ( )` function requires 4 arguments.

The arguments are specified as follows:

- lines** the maximum vertical dimension of the new window, specified in units of lines
- cols** the maximum horizontal dimension of the new window specified in units of columns
- begy** the line coordinate, specifying where the new window will start in relation to the stdscr vertical dimension
- begx** the column coordinate, specifying where the new window will start in relation to the stdscr horizontal dimension.

`newwin( )` computes the dimension of the new window by using the following equations:

```
if ( begy + lines > LINES )
    lines = LINES - begy;
if ( begx + cols > COLS )
    cols = COLS - begx;
if ( lines == 0 )
    lines = LINES - begy;
if ( cols == 0 )
    cols = COLS - begx;
```

The dimensions of the new window must be equal to or within the maximum dimensions of the `stdscr` window, which is specified in the variables `LINES` and `COLS`.

## *Adding characters to a window*

The most primitive way to add a character to a window is by using `waddch(win,ch)`. This function is used by all curses functions which add characters to a window in some way. We could say that `waddch()` is the lowest level curses function used to add a character to a window.

In keeping with the philosophy of UNIX, that is , build on the work of others, the standard I/O package function `printf()` has been implemented in the curses package as `wprintw()`.

## Functions that add characters/prints string to a window:

```
int waddch(win, ch)           puts char in stdscr
int addch(ch)
int mvaddch(y, x, ch)
int mvwaddch(win, y, x, ch)

int waddstr(win, str)        puts string in stdscr
int addstr(str)
int mvaddstr(y, x, str)
int mvwaddstr(win, y, x, str)

int mvprintw(y, x, fmt, args)  move/print string in stdscr

int mvwprintw(win, x, y, fmt, args) move/print string in a window

int wprintw(win, fmt, args)   print string in a window
```

## *Moving around a window*

To move to a new position within a window you use the `wmove(win,y,x)` function. The new coordinates must be within the bounds of the window's maximum dimensions or `ERR` is returned and the coordinates are not changed.

## *Setting Terminal Input Modes*

In normal operations, the terminal operates in full-duplex mode – meaning the characters that are typed at the keyboard are echoed back to the terminal via the computer.

Characters may be typed at any time, even if output is still occurring. The `tty` device driver buffers input characters as they are received in a system character input buffer (`clist`).

However, certain characters received by the driver have special meaning, and the driver interprets them depending on how it is configured. They are as follows:

**Interrupt** (ASCII DEL) Generates an interrupt signal which is received by all running processes associated with the terminal. A process receiving this signal terminates, unless it has been programmed to ignore it.

**Quit** (Control-|) Identical to **Interrupt**, except that a file is created in the current directory containing a core image of the process.

**Stop** (Control-S) Temporarily suspends output to the terminal.

- Start** (Control-Q) Resumes output currently suspended by **Stop**.
- Erase** ( # ) erases the last character typed, but will not erase beyond the start of a line, as delimited by a NL, EOL, or EOF character. This is often set to Control-H.
- Kill** ( @ ) Deletes the entire line, as delimited by a NL, EOF, or EOL character. This is often set to Control-C or Control-U.
- New-Line** (ASCII NL) Line delimiter (satisfies the read). Characters stored in the *tty* driver's internal buffer are passed over to the process which is reading from the terminal and the buffer is flushed.

\* Curses retains these terminal modes by default.

*Curses* provides a comprehensive set of routines for setting and reading the terminal driver modes. Some of these functions are summarized below:

**cbreak( )** Sets the terminal into cbreak mode. this means that characters typed at the terminal are not buffered by the tty driver; they become immediately available to the program as they are entered at the keyboard, without having to wait for a line delimiter.

**raw ( )** The terminal is set into raw mode. This works the same way cbreak does, except that all characters are passed directly to the program without being interpreted. This includes interrupt, quit, start and stop characters.

**echo ( )** used in conjunction with `wgetch( )`. If `echo` is enabled, then `wgetch( )` automatically adds character typed at the keyboard to the working window. When the window is refreshed the characters read by `wgetch( )` are sent back to the terminal.

**noecho( )** In this mode, characters are not added to the window by `wgetch( )`. It is up to you to add them to it. This is normally done with `waddch( )` followed by `wrefresh( )`. This mode is often required by programs which have to do their own echoing, possibly in a controlled area of the screen, or if the characters entered are control characters specific to the program, specifying some action. This type of character is often not displayed at all.

**savetty( )** This function saves the settings of the current terminal (*tty*) modes in the internal buffer. It is automatically called by `initscr( )`.

**resetty( )** This function restores the terminal (*tty*) modes to what was originally saved in the internal buffer, set by `savetty( )`.

### *Reading from the Keyboard*

The function `wgetch(win)` is used to get keyboard input from a specified window. Depending on what input modes are set, `wgetch()` returns the key typed at the keyboard.

## Functions that read input:

<code>int wgetch(win)</code>	get char via a window
<code>int getch()</code>	
<code>int mvgetch(y,x)</code>	
<code>int mvwgetch(win,y,x)</code>	
<code>int wgetstr(win,str)</code>	get string via window to a buffer
<code>int getstr(str)</code>	
<code>int mvgetstr(y,x,str)</code>	
<code>int mvwgetstr(win,y,x,str)</code>	
<code>int scanw(fmt,args)</code>	get values via stdscr
<code>int mvscanw(y,x,fmt,args)</code>	move & get values via stdscr
<code>int mvwscanw(win,y,x,fmt,args)</code>	move & get values via a window

## *Reading from a Window*

The only method of reading a character from a window is to use the pseudo-function `winch(win)` (`inch()` for `stdscr`). This function returns the character under the cursor at the current  $y,x$  coordinates of a window, although this character may contain attributes if they are set, as well as the character itself.

Some functions:

```
int winch(win)           get char at window cursor
int inch()
int mvinch(y,x)
int mvwinch(win,y,x)
```

```
int wansch(win, ch)           insert character in a window
int insch(ch)
int mvansch(y, x, ch)
int mvwansch(win, y, x, ch)
```

```
int winsertln(win)           insert new line in a window
int insertln()
int mvinsertln(y, x)
int mvwinsertln(win, y, x)
```

### *Deleting characters in a Window*

The function `wdelch(win)` is used to delete a character under the cursor on the specified window. Each character along the current line after the cursor in the window is shifted left one position and the last character becomes blank. The current `y,x` coordinates remain unchanged.

## Some functions:

<code>int wdelch(win)</code>	delete a char in a window
<code>int delch()</code>	
<code>int mvdelch(y,x)</code>	
<code>int mvwdelch(win,y,x)</code>	
<code>int wdeleteln(win)</code>	delete a line in a window
<code>int deleteln()</code>	
<code>int mvdeleteln(y,x)</code>	
<code>int mvwdeleteln(win,y,x)</code>	

## *Deleting a Window*

The process of deleting a window loses the contents of it and all space associated with it is freed up. The function which does this is `delwin(win)`. If the window has any associated sub-windows they should be deleted first because `delwin( )` does not delete sub-windows automatically, although by deleting the parent window it effectively invalidates the sub-window. It is important to do this so that memory is freed up for other uses. Also, if you access a sub-window after the parent is deleted, your program may dump core which will almost definitely leave the terminal in an unpredictable state.

## *Relocating a Window*

The function `mvwin(win,begy,begx)` is used to relocate a window. The given window is moved to the new location as specified by `begy` and `begx`, whose coordinates are the window's new top left-hand corner coordinates. The move must not allow the window or portion of it to overlap the terminal screen; if it does, the window is not moved and the function returns `ERR`. When a window is moved from one location to another, the internal `WINDOW` data structure remains intact except, of course, `_begy` and `_begx`. This means that characters displayed with attributes will retain them throughout the move.

## REFERENCES

Goodheart, Berny. UNIX Curses Explained. Australia: Prentice Hall, 1991.

Ben-Halim, Zeyd M. and Raymond, Eric S.

<http://web.cs.mun.ca/~rod/ncurses/ncurses.html#scope>

Dickey, Thomas. <http://dickey.his.com/ncurses/ncurses.faq.html>

<http://dickey.his.com/ncurses/ncurses.html>

Matloff, Norman.

<http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/Curses.html>

Padala, Pradeep. [http://www.cise.ufl.edu/~ppadala/ncurses/NCURSES\\_HOWTO/](http://www.cise.ufl.edu/~ppadala/ncurses/NCURSES_HOWTO/)

<http://www.linuxdoc.org/HOWTO/NCURSES-Programming-HOWTO/tools.html>

<http://www.cs.vu.nl/pub/minix/2.0.0/wwwman/man3/curses.3.html>

<http://www.vexus.ca/CDK.html>