

NAME

screen.h – a simple screen manipulation library for GNU/Linux and other UNIX-like systems

SYNOPSIS

```
#include <screen.h>
```

DESCRIPTION

screen.h is an implementation of the ANSI escape sequences that were used on VT100 and compatible/subsequent terminals, with which most modern UNIX/Linux terminals are compatible. However, screen.h isn't something that I would recommend for 'real,' large programming projects, because the way it does what it does is rather 'quick and dirty.'

screen.h has been successfully tested on Linux 2.6 and OpenBSD 4.7. It won't, of course, work on Windows - the software will work, the escape sequences will be visible, not their effects. It has not been tested on DOS.

The library evolved from a crude header file containing sequences to change on-screen colours, as part of another project.

The screen.h library is stand-alone and self-sufficient, containing all of its own functions. However, of course, stdio.h still needs to be #included.

screen.h is compatible with C++ as well as C, and any compiler that compiles Linux binaries from C and C++ code should be able to use screen.h. This is because the code in screen.h manipulates the terminal, so it doesn't use any special features in any dialect of C or C++.

EXAMPLES**To change text colour:**

```
printf("%s", red); /* change the colour to red */
printf("%s", fg_default); /* change the colour back to normal */
```

To change background colour:

```
printf("%s", bg_green); /* change the colour to green */
printf("%s", bg_default); /* change the colour back to normal */
```

To set fore- and background colours together:

```
setcolours(33, 44); /* foreground yellow, background blue */
```

It doesn't matter which way around you set the colours, because each numeric code is an escape number - whichever one is transmitted first will take effect, the the other. Below is a list of the different colours:

(form: foreground, background, colour)

```
30, 40, black
31, 41, red
32, 42, green
33, 43, yellow
34, 44, blue
35, 45, purple
36, 46, cyan
37, 47, white
38, 48, default
```

To change/revert text attributes: Note: faint() and invis() can only be reversed by passing normalscr().

```
faint(); /* makes text appear faint */
invis(); /* makes text invisible */
```

```
bold(1); /* embolden following text */
bold(0); /* end bold */
```

```
standout(1); /* make following text stand out */
```

```
standout(0); /* end stand out */
```

```
underline(1); /* underline text */
underline(0); /* end underline */
```

```
blink(1); /* make text blink */
blink(0); /* end blink */
```

```
normalscr(); /* return all settings to normal */
```

To clear the screen:

```
clearscr();
or
printf("%s", clear);
```

To move the cursor to a different position on the screen:

```
move_up(5); /* to go up 5 rows */
move_down(3); /* to go down 3 rows */
move_right(8); /* to go right 8 columns */
move_left(12); /* to go left 12 columns */
move_xy(12, 30); /* go go to column 12, row 30 */
```

To show one character at a specific position on the screen:

```
show_xy('@', 10, 10); /* show '@' at column 10, row 10 */
```

To save/restore the cursor's position: Note: saving the cursor's position also saves its attributes, such as boldness, underline, %c, and the fore- and background colours as well. Likewise, restoring also restores all of the saved cursor's attributes.

```
sav_cur_pos(); /* saves the cursor's position */
res_cur_pos(); /* restores the cursor to saved position */
```

TUTORIAL

In this manual, I'm going to give a short tutorial, in the form of a small program and an explanation, of how to use the functions defined in this library. It isn't intended to patronize the reader, but make sure that I have been clear enough in my explanations of the functions.

1 – Changing the colour of a piece or text.

Changing the colour of a the text on the screen is, so far, the only thing that isn't covered by a function, other than the *setcolours()* command. This isn't because I've been too lazy to write such a series of functions, but because I don't yet know how to do what I want to do, which is to write my own implementation of *printf()*, but which displays the text that is passed to it in a specific colour, for example by having a *red()* command, which acts the same as *printf()*, but in red, &c.

The first example I'm going to give is that - a brief 'hello world' program, using colour (try it yourself: colour1.c):

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

int main() {
    printf("%s", red);
    printf(" Hello, world!\n");
    printf("%s", fg_default);
    return 0; }
```

This is just your basic 'hello world' program, but with two extra *printf()* statements; one before the message, and one after. *printf("%s", red)* displays the escape sequence defined in screen.h to turn the text red, and *printf("%s", fg_default)* turns the text back to what it normally is - normally light grey.

You can, by changing those two lines of code that added the colour, change the background colour as well

(colour2.c):

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

int main() {
    printf("%s%s", yellow, bg_blue);
    printf(" Hello, world!\n");
    printf("%s", normal);
    return 0; }
```

As you can see, there are now two strings being printed in the first *printf()* statement, which turn the foreground yellow, and the background blue. The last *printf()* statement only uses one string, like the previous example, because, since it is setting both the foreground and the background back to normal, it doesn't need to do them separately: there is a single escape sequence which will remove all text attributes, including colour.

However, I have written two functions which will do these things without you needing to type *printf()* every time. See this next example (colour3.c):

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

int main() {
    setcolours(33, 44);
    printf(" Hello, world!\n");
    normalscr();
    return 0; }
```

The command *setcolours()* sets the colours of the foreground and of the background together, using the colour codes. You can do it either way round (by passing (33, 44) or (44, 33)). There is a list of all the colour codes in the previous chapter (EXAMPLES - To set fore- and background colours together). The *normalscr()* command prints the 'normal' string, meaning that you don't actually need to type *printf("%s", normal)* when you want to remove all the text's attributes.

2 - Changing the attributes of a piece of text.

On a video terminal (or xterm/rxvt/gnome-terminal/konsole/whatever), you can change attributes of the text, to make it glow brighter, or be underlined, or other similar things.

A simple program to display a message in bold would be this (bold.c):

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

int main() {
    bold(1);
    printf(" This text is bold.\n");
    bold(0);
    return 0; }
```

The code in the function called *bold()* will print one of two escape sequences, depending on what input is supplied. If you call it with *bold(1)*, it will make the text following it bold, and if you call it with *bold(0)*, it will make the text following it not bold.

There are three other functions in the library, which are used in the exact same way: *standout()*, which makes text stand out better, *underline()*, which underlines text, and *blink()*, which makes the text blink on and off, whatever colour it already is. The code would look nearly identical for each function; to try it out, copy the file bold.c to three others: *standout.c*, *underline.c*, and *blink.c*, and replace *bold* with whatever the

filename is.

There are two functions which aren't directly reversible: *faint()* and *invis()*, which make the text appear fainter, and disappear completely, respectively. To use them, you call them without any arguments. Try making a new file, *faint.c*:

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

int main() {
    faint();
    printf(" This text is faint.\n");
    return 0; }
```

and run it. Then make a copy of that file to *invis.c*, and replace *faint()*; with *invis()*; , and then try that.

However, since those two functions aren't directly reversible, there has to be another way of reversing them. The function *normalscr()* will do this, however it will also set colours to normal. Make a new file called *ftn-inv.c* and put the following code into it:

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

int main() {
    faint();
    printf(" This text is faint.\n");
    normalscr();
    invis();
    printf(" This text is invisible.\n");
    normalscr();
    printf(" This text is normal.\n");
    return 0; }
```

Of course, you aren't expecting to see "This line is invisible." on the screen, but "This text is faint." should appear two lines above "This text is normal." but fainter.

3 - Moving around on the screen.

As well as being able to control the screen's colours and attributes of the text, you can also move the cursor around on the screen, and clear it completely.

Create a new file, call it *clear.c*, and type the following code into it:

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

int main() {
    clearscr();
    return 0; }
```

This program will clear the screen, by first scrolling the entire contents of the screen as it is now above the first line, and then relocating the cursor to row 1, column 1.

For whatever reason, you may need to move the cursor around by one or more characters at a time. This is where the *move_up()*, *move_down()*, *move_right()*, and *move_left()* functions are useful. Copy this code to a file called *box.c*:

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

```

#include <screen.h>

int main() {
    int i;
    clearscr();
    for(i = 1; i <= 20; ++i) {
        printf("*");
        move_right(1); }
    newline();
    for(i = 1; i <= 3; ++i) {
        printf("*");
        move_right(19);
        printf("*");
        newline(); }
    move_up(3);
    move_right(4);
    printf("message");
    move_down(5);
    newline();
    return 0; }

void newline() {
    move_left(80);
    move_down(1);
    return; }

```

This code should draw a box using stars, and display the word 'message' in the middle of it. Of course, if you were writing a real program, you would use a newline escape (\n) - but I just wanted to demonstrate all four functions.

You can also write programs with the capability to move the cursor to a specific point on the screen, similar to Pascal's *gotoxy* and QuickBASIC's *LOCATE* statements. This is done using a function called *move_xy()*, which accepts its input as two on-screen coordinates. Try this program (moxxy.c):

```

#include <stdio.h>
#include <screen.h>

int main() {
    move_xy(5, 5);
    printf("Hello.");
    move_xy(1, 15);
    return 0; }

```

The first *move_xy()* statement puts the cursor at column 5, row 5, which is where the message "Hello." is started from, and the second puts the cursor at column 1, row 15, where the shell reappears once the program has terminated.

To print a single character at a specific coordinate, you can use the *show_xy()* function, which prints a single character at the specified coordinates. A simple program that prints a '*' to a pair of coordinates would look like this (showxy.c):

```

#include <stdio.h>
#include <screen.h>

int main() {
    show_xy("*", 5, 5);
}

```

```
    move_xy(1, 7);
    return 0; }
```

The *show_xy()* in that program will print a '*' character at column 5, row 5. The *move_xy()* is there to make sure that when the program finishes, the prompt is put at the start of a newline.

Another pair of functions which might be useful when moving around the screen is *sav_cur_pos()* and *res_cur_pos()* which save and restore the cursor's position on the screen. A simple demonstration of these functions would be this small program (savres.c):

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

int main() {
    clrscr();
    sav_cur_pos();
    move_xy(50, 10);
    printf("Hello, there.");
    res_ruc_pos();
    return 0; }
```

In which the screen is cleared, moving the cursor to column 1, row 1, and then immediately *sav_cur_pos()* saves the position of the cursor to memory. *move_xy()* then moves the cursor to column 50, row 10, and a message is printed. Then *res_cur_pos()* retrieves the position of the cursor as it was when it was saved, and replaces the cursor at that position.

FILES

/usr/include/screen.h – The library should be installed here. That way, you can include it with `<screen.h>`, instead of "path/to/screen.h".

HISTORY

Note: I don't believe in version numbers the way other people use them. Each time I add a new feature I increment the version number, so that I can recall which version has which features.

- 1 – used pure ANSI escapes (^[...), and used no actual functions.
- 2 – replaced pure ANSI escapes with C ones (\e ...).
- 3 – added all ANSI escape sequences.
- 4 – added functions implementing on-screen movement/relocation.
- 5 – added functions to clear the terminal, change colours, and alter text attributes.

AUTHOR

Mark Tuson (markftuson@gmail.com)

SEE ALSO

The GNU General Public License - Free Software Foundation (<http://www.gnu.org/licenses/gpl.html>)

The C Programming Language - Brian Kernighan & Dennis Ritchie (<http://plan9.bell-labs.com/cm/cs/cbook/>)