libtcod Documentation

Release 1.8.2

Richard Tew

Oct 23, 2018

Core

CHAPTER 1

Colors

libtcod uses 32-bit color, therefore your OS desktop must also use 32-bit color. A color is defined by its red, green and blue component between 0 and 255.

You can use the following predefined colors (hover over a color to see its full name and R,G,B values):

INSERT COLOUR TABLE IN A PAINLESS MANNER

1.1 Create your own colors

You can create your own colours using a set of constructors, both for RGB and HSV values.

```
/* RGB */
TCOD_color_t my_color= { 24, 64, 255 };
TCOD_color_t my_other_color = TCOD_color_RGB(24, 64, 255);
/* HSV */
TCOD_color_t my_yet_another_color = TCOD_color_HSV(321.0f, 0.7f, 1.0f);
```

```
// RGB
TCODColor myColor(24, 64, 255);
// HSV
TCODColor myOtherColor(321.0f, 0.7f, 1.0f);
```

```
my_color = libtcod.Color(24, 64, 255)
```

1.2 Compare two colors

bool **TCOD_color_equals** (TCOD_color_t *c1*, TCOD_color_t *c2*)
Return a true value if c1 and c2 are equal.

1.3 Add and subtract Colors

TCOD_color_t **TCOD_color_add**(TCOD_color_t *c1*, TCOD_color_t *c2*)
Add two colors together and return the result.

Return A new TCOD_color_t struct with the result.

Parameters

- `c1`: The first color.
- `c2`: The second color.

TCOD_color_t **TCOD_color_subtract**(TCOD_color_t *c1*, TCOD_color_t *c2*)
Subtract c2 from c1 and return the result.

Return A new TCOD_color_t struct with the result.

Parameters

- `c1`: The first color.
- `c2`: The second color.

1.4 Multiply Colors together

TCOD_color_t **TCOD_color_multiply**(TCOD_color_t *c1*, TCOD_color_t *c2*)
Multiply two colors together and return the result.

Return A new TCOD_color_t struct with the result.

Parameters

- `c1`: The first color.
- `c2`: The second color.

TCOD_color_t **TCOD_color_multiply_scalar**(TCOD_color_t *c1*, float *value*)
Multiply a color with a scalar value and return the result.

Return A new TCOD_color_t struct with the result.

Parameters

- `c1`: The color to multiply.
- `value`: The scalar float.

1.5 Interpolate between two colors

TCOD_color_t **TCOD_color_lerp**(TCOD_color_t *c1*, TCOD_color_t *c2*, float *coef*)
Interpolate two colors together and return the result.

Return A new TCOD_color_t struct with the result.

Parameters

- `c1`: The first color (where coef if 0)
- `c2`: The second color (where coef if 1)
- `coef`: The coefficient.

1.6 Define a color by its hue, saturation and value

After this function is called, the r,g,b fields of the color are calculated according to the h,s,v parameters.

void **TCOD_color_set_HSV**(TCOD_color_t *color, float *hue*, float *saturation*, float *value*)
Sets a colors values from HSV values.

Parameters

- `color`: The color to be changed.
- `hue`: The colors hue (in degrees.)
- `saturation`: The colors saturation (from 0 to 1)
- `value`: The colors value (from 0 to 1)

These functions set only a single component in the HSV color space.

void **TCOD_color_set_hue**(TCOD_color_t *color, float *hue*)
Change a colors hue.

Parameters

- `color`: Pointer to a color struct.
- `hue`: The hue in degrees.

void **TCOD_color_set_saturation**(TCOD_color_t *color, float *saturation*)
Change a colors saturation.

Parameters

- `color`: Pointer to a color struct.
- `saturation`: The desired saturation value.

void **TCOD_color_set_value**(TCOD_color_t *color, float *value*)
Change a colors value.

Parameters

- `color`: Pointer to a color struct.
- `value`: The desired value.

1.7 Get a color hue, saturation and value components

void **TCOD_color_get_HSV**(TCOD_color_t *color*, float **hue*, float **saturation*, float **value*)
Get a set of HSV values from a color.

The hue, saturation, and value parameters can not be NULL pointers,

Parameters

- `color`: The color
- `hue`: Pointer to a float, filled with the hue. (degrees)
- `saturation`: Pointer to a float, filled with the saturation. (0 to 1)
- `value`: Pointer to a float, filled with the value. (0 to 1)

Should you need to extract only one of the HSV components, these functions are what you should call. Note that if you need all three values, it's way less burdensome for the CPU to call *TCODColor::getHSV()*.

float **TCOD_color_get_hue** (TCOD_color_t *color*)
Return a colors hue.

Return The colors hue. (degrees)

Parameters

- `color`: A color struct.

float **TCOD_color_get_saturation** (TCOD_color_t *color*)
Return a colors saturation.

Return The colors saturation. (0 to 1)

Parameters

- `color`: A color struct.

float **TCOD_color_get_value** (TCOD_color_t *color*)
Get a colors value.

Return The colors value. (0 to 1)

Parameters

- `color`: A color struct.

1.8 Shift a color's hue up or down

The hue shift value is the number of grades the color's hue will be shifted. The value can be negative for shift left, or positive for shift right. Resulting values H < 0 and H >= 360 are handled automatically.

void **TCOD_color_shift_hue** (TCOD_color_t **color*, float *hshift*)
Shift a colors hue by an amount.

Parameters

- `color`: Pointer to a color struct.
- `hue_shift`: The distance to shift the hue, in degrees.

1.9 Scale a color's saturation and value

void **TCOD_color_scale_HSV** (TCOD_color_t **color*, float *saturation_coef*, float *value_coef*)
Scale a colors saturation and value.

Parameters

- `color`: Pointer to a color struct.
- `saturation_coef`: Multiplier for this colors saturation.
- `value_coef`: Multiplier for this colors value.

1.10 Generate a smooth color map

You can define a color map from an array of color keys. Colors will be interpolated between the keys. 0 -> black 4 -> red 8 -> white Result:

INSERT TABLE.

void **`TCOD_color_gen_map`** (TCOD_color_t **map*, int *nb_key*, **`const`** TCOD_color_t **key_color*, **`const`** int **key_index*)

Generate an interpolated gradient of colors.

```
TCOD_color_t[256] gradient;
TCOD_color_t[4] key_color = {TCOD_black, TCOD_dark_amber,
                             TCOD_cyan, TCOD_white};
int[4] key_index = {0, 64, 192, 255};
TCOD_color_gen_map(&gradient, 4, &key_color, &key_index);
```

Parameters

- `map`: Array to fill with the new gradient.
- `nb_key`: The array size of the key_color and key_index parameters.
- `key_color`: An array of colors to use, in order.
- `key_index`: An array mapping key_color items to the map array.

CHAPTER 2

Console

2.1 Initializing the console

2.1.1 Creating the game window

enum TCOD_renderer_t
The available renderers.

Values:

TCOD_RENDERER_GLSL
An OpenGL implementation using a shader.

TCOD_RENDERER_OPENGL
An OpenGL implementation without a shader.

Performs worse than TCOD_RENDERER_GLSL without many benefits.

TCOD_RENDERER_SDL
A software based renderer.

The font file is loaded into RAM instead of VRAM in this implementation.

TCOD_RENDERER_SDL2
A new SDL2 renderer.

Allows the window to be resized. New in version 1.8.

TCOD_RENDERER_OPENGL2
A new OpenGL 2.1 renderer.

Allows the window to be resized. New in version 1.9.

TCOD_NB_RENDERERS

void **TCOD_console_init_root** (int *w*, int *h*, **const** char **title*, bool *fullscreen*, *TCOD_renderer_t renderer*)
Initialize the libtcod graphical engine.

You may want to call TCOD_console_set_custom_font BEFORE calling this function. By default this function loads libtcod's `terminal.png` image from the working directory.

Parameters

- `w`: The width in tiles.
- `h`: The height in tiles.
- `title`: The title for the window.
- `fullscreen`: Fullscreen option.
- `renderer`: Which renderer to use when rendering the console.

Afterwards TCOD_quit must be called before the program exits.

void **TCOD_quit**(void)
Shutdown libtcod.

This must be called before your program exits. New in version 1.8.

2.1.2 Using a custom bitmap font

enum TCOD_font_flags_t
These font flags can be OR'd together into a bit-field and passed to TCOD_console_set_custom_font.

Values:

TCOD_FONT_LAYOUT_ASCII_INCOL =1
Tiles are arranged in column-major order.

0 3 6 1 4 7 2 5 8

TCOD_FONT_LAYOUT_ASCII_INROW =2
Tiles are arranged in row-major order.

0 1 2 3 4 5 6 7 8

TCOD_FONT_TYPE_GREYSCALE =4
Converts all tiles into a monochrome gradient.

TCOD_FONT_TYPE_GRAYSCALE =4

TCOD_FONT_LAYOUT_TCOD =8
A unique layout used by some of libtcod's fonts.

void **TCOD_console_set_custom_font**(**const** char **fontFile*, int *flags*, int *nb_char_horiz*, int *nb_char_vertic*)
Set a font image to be loaded during initialization.

`fontFile` will be case-sensitive depending on the platform.

Parameters

- `fontFile`: The path to a font image.
- `flags`: A TCOD_font_flags_t bit-field describing the font image contents.
- `nb_char_horiz`: The number of columns in the font image.
- `nb_char_vertic`: The number of rows in the font image.

2.1.3 Using custom characters mapping

void **TCOD_console_map_ascii_code_to_font**(int *asciiCode*, int *fontCharX*, int *fontCharY*)
Remap a character code to a tile.

X,Y parameters are the coordinate of the tile, not pixel-coordinates.

Parameters

- `asciiCode`: Character code to modify.
- `fontCharX`: X tile-coordinate, starting from the left at zero.
- `fontCharY`: Y tile-coordinate, starting from the top at zero.

void **TCOD_console_map_ascii_codes_to_font**(int *asciiCode*, int *nbCodes*, int *fontCharX*, int *fontCharY*)
Remap a series of character codes to a row of tiles.

This function always assigns tiles in row-major order, even if the TCOD_FONT_LAYOUT_ASCII_INCOL flag was set.

Parameters

- `asciiCode`: The starting character code.
- `nbCodes`: Number of character codes to assign.
- `fontCharX`: First X tile-coordinate, starting from the left at zero.
- `fontCharY`: First Y tile-coordinate, starting from the top at zero.

void **TCOD_console_map_string_to_font**(**const** char **s*, int *fontCharX*, int *fontCharY*)
Remap a string of character codes to a row of tiles.

This function always assigns tiles in row-major order, even if the TCOD_FONT_LAYOUT_ASCII_INCOL flag was set.

Parameters

- `s`: A null-terminated string.
- `fontCharX`: First X tile-coordinate, starting from the left at zero.
- `fontCharY`: First Y tile-coordinate, starting from the top at zero.

2.1.4 Fullscreen mode

void **TCOD_console_set_fullscreen**(bool *fullscreen*)
Set the display to be full-screen or windowed.

Parameters

- `fullscreen`: If true the display will go full-screen.

bool **TCOD_console_is_fullscreen**(void)
Return true if the display is full-screen.

2.1.5 Communicate with the window manager

bool **TCOD_console_is_active**(void)
Return true if the window has keyboard focus.

bool **TCOD_console_has_mouse_focus**(void)
Return true if the window has mouse focus.

bool **TCOD_console_is_window_closed**(void)
Return true if the window is closing.

void **TCOD_console_set_window_title**(**const** char **title*)
Change the title string of the active window.

Parameters

- `title`: A utf8 string.

2.1.6 libtcod's Credits

void **TCOD_console_credits**(void)

void **TCOD_console_credits_reset**(void)

bool **TCOD_console_credits_render**(int *x*, int *y*, bool *alpha*)

2.2 Drawing on the root console

2.2.1 Basic printing functions

void **TCOD_console_set_default_foreground**(TCOD_console_t *con*, TCOD_color_t *col*)

void **TCOD_console_set_default_background**(TCOD_console_t *con*, TCOD_color_t *col*)

void **TCOD_console_set_background_flag**(TCOD_console_t *con*, *TCOD_bkgnd_flag_t flag*)
Set a consoles default background flag.

Parameters

- `con`: A console pointer.
- `flag`: One of `TCOD_bkgnd_flag_t`.

void **TCOD_console_clear**(TCOD_console_t *con*)
Clear a console to its default colors and the space character code.

void **TCOD_console_put_char**(TCOD_console_t *con*, int *x*, int *y*, int *c*, *TCOD_bkgnd_flag_t flag*)
Draw a character on a console using the default colors.

Parameters

- `con`: A console pointer.
- `x`: The X coordinate, the left-most position being 0.
- `y`: The Y coordinate, the top-most position being 0.
- `c`: The character code to place.

- `flag`: A TCOD_bkgnd_flag_t flag.

void **TCOD_console_put_char_ex**(TCOD_console_t *con*, int *x*, int *y*, int *c*, TCOD_color_t *fore*, TCOD_color_t *back*)
Draw a character on the console with the given colors.

Parameters

- `con`: A console pointer.
- `x`: The X coordinate, the left-most position being 0.
- `y`: The Y coordinate, the top-most position being 0.
- `c`: The character code to place.
- `fore`: The foreground color.
- `back`: The background color. This color will not be blended.

void **TCOD_console_set_char**(TCOD_console_t *con*, int *x*, int *y*, int *c*)
Change a character on a console tile, without changing its colors.

Parameters

- `con`: A console pointer.
- `x`: The X coordinate, the left-most position being 0.
- `y`: The Y coordinate, the top-most position being 0.
- `c`: The character code to set.

void **TCOD_console_set_char_foreground**(TCOD_console_t *con*, int *x*, int *y*, TCOD_color_t *col*)
Change the foreground color of a console tile.

Parameters

- `con`: A console pointer.
- `x`: The X coordinate, the left-most position being 0.
- `y`: The Y coordinate, the top-most position being 0.
- `col`: The foreground color to set.

void **TCOD_console_set_char_background**(TCOD_console_t *con*, int *x*, int *y*, TCOD_color_t *col*, *TCOD_bkgnd_flag_t* *flag*)
Blend a background color onto a console tile.

Parameters

- `con`: A console pointer.
- `x`: The X coordinate, the left-most position being 0.
- `y`: The Y coordinate, the top-most position being 0.
- `col`: The background color to blend.
- `flag`: The blend mode to use.

void **TCOD_console_rect**(TCOD_console_t *con*, int *x*, int *y*, int *w*, int *h*, bool *clear*, *TCOD_bkgnd_flag_t flag*)

Draw a rectangle onto a console.

Parameters

- `con`: A console pointer.
- `x`: The starting region, the left-most position being 0.
- `y`: The starting region, the top-most position being 0.
- `rw`: The width of the rectangle.
- `rh`: The height of the rectangle.
- `clear`: If true the drawing region will be filled with spaces.
- `flag`: The blending flag to use.

void **TCOD_console_hline**(TCOD_console_t *con*, int *x*, int *y*, int *l*, *TCOD_bkgnd_flag_t flag*)

Draw a horizontal line using the default colors.

This function makes assumptions about the fonts character encoding. It will fail if the font encoding is not `cp437`.

Parameters

- `con`: A console pointer.
- `x`: The starting X coordinate, the left-most position being 0.
- `y`: The starting Y coordinate, the top-most position being 0.
- `l`: The width of the line.
- `flag`: The blending flag.

void **TCOD_console_vline**(TCOD_console_t *con*, int *x*, int *y*, int *l*, *TCOD_bkgnd_flag_t flag*)

Draw a vertical line using the default colors.

This function makes assumptions about the fonts character encoding. It will fail if the font encoding is not `cp437`.

Parameters

- `con`: A console pointer.
- `x`: The starting X coordinate, the left-most position being 0.
- `y`: The starting Y coordinate, the top-most position being 0.
- `l`: The height of the line.
- `flag`: The blending flag.

void **TCOD_console_print_frame**(TCOD_console_t *con*, int *x*, int *y*, int *w*, int *h*, bool *empty*, *TCOD_bkgnd_flag_t flag*, **const** char **fmt*, ...)

2.2.2 Background effect flags

enum TCOD_bkgnd_flag_t

Values:

TCOD_BKGND_NONE

TCOD_BKGND_SET

TCOD_BKGND_MULTIPLY

TCOD_BKGND_LIGHTEN

TCOD_BKGND_DARKEN

TCOD_BKGND_SCREEN

TCOD_BKGND_COLOR_DODGE

TCOD_BKGND_COLOR_BURN

TCOD_BKGND_ADD

TCOD_BKGND_ADDA

TCOD_BKGND_BURN

TCOD_BKGND_OVERLAY

TCOD_BKGND_ALPH

TCOD_BKGND_DEFAULT

2.2.3 String printing alignment

enum TCOD_alignment_t
Print justification options.

Values:

TCOD_LEFT

TCOD_RIGHT

TCOD_CENTER

void **TCOD_console_set_alignment**(TCOD_console_t *con*, *TCOD_alignment_t alignment*)
Set a consoles default alignment.

Parameters

- `con`: A console pointer.
- `alignment`: One of TCOD_alignment_t

TCOD_alignment_t **TCOD_console_get_alignment**(TCOD_console_t *con*)
Return a consoles default alignment.

2.2.4 Printing functions using 8-bit encodings

void **TCOD_console_print**(TCOD_Console **con*, int *x*, int *y*, **const** char **fmt*, ...)
Print a string on a console, using default colors and alignment.

Parameters

- `con`: A console pointer.
- `x`: The starting X coordinate, the left-most position being 0.
- `y`: The starting Y coordinate, the top-most position being 0.

- `fmt`: A format string as if passed to printf.
- `...`: Variadic arguments as if passed to printf.

void **TCOD_console_print_ex**(TCOD_Console *con, int x, int y, TCOD_bkgnd_flag_t flag, TCOD_alignment_t alignment, **const** char *fmt, ...)

Print a string on a console, using default colors.

Parameters

- `con`: A console pointer.
- `x`: The starting X coordinate, the left-most position being 0.
- `y`: The starting Y coordinate, the top-most position being 0.
- `flag`: The blending flag.
- `alignment`: The font alignment to use.
- `fmt`: A format string as if passed to printf.
- `...`: Variadic arguments as if passed to printf.

int **TCOD_console_print_rect**(TCOD_Console *con, int x, int y, int w, int h, **const** char *fmt, ...)

Print a string on a console constrained to a rectangle, using default colors and alignment.

Return The number of lines actually printed.

Parameters

- `con`: A console pointer.
- `x`: The starting X coordinate, the left-most position being 0.
- `y`: The starting Y coordinate, the top-most position being 0.
- `w`: The width of the region. If 0 then the maximum width will be used.
- `h`: The height of the region. If 0 then the maximum height will be used.
- `fmt`: A format string as if passed to printf.
- `...`: Variadic arguments as if passed to printf.

int **TCOD_console_print_rect_ex**(TCOD_Console *con, int x, int y, int w, int h, TCOD_bkgnd_flag_t flag, TCOD_alignment_t alignment, **const** char *fmt, ...)

Print a string on a console constrained to a rectangle, using default colors.

Return The number of lines actually printed.

Parameters

- `con`: A console pointer.
- `x`: The starting X coordinate, the left-most position being 0.
- `y`: The starting Y coordinate, the top-most position being 0.
- `w`: The width of the region. If 0 then the maximum width will be used.
- `h`: The height of the region. If 0 then the maximum height will be used.
- `flag`: The blending flag.
- `alignment`: The font alignment to use.

- `fmt`: A format string as if passed to printf.
- `...`: Variadic arguments as if passed to printf.

int **TCOD_console_get_height_rect**(TCOD_Console *con, int x, int y, int w, int h, **const** char *fmt, ...)

Return the number of lines that would be printed by the.

Return The number of lines that would have been printed.

Parameters

- `con`: A console pointer.
- `x`: The starting X coordinate, the left-most position being 0.
- `y`: The starting Y coordinate, the top-most position being 0.
- `w`: The width of the region. If 0 then the maximum width will be used.
- `h`: The height of the region. If 0 then the maximum height will be used.
- `fmt`: A format string as if passed to printf.
- `...`: Variadic arguments as if passed to printf.

2.2.5 Printing functions using UTF-8

void **TCOD_console_printf**(TCOD_Console *con, int x, int y, **const** char *fmt, ...)

Format and print a UTF-8 string to a console.

New in version 1.8.

void **TCOD_console_printf_ex**(TCOD_Console *con, int x, int y, *TCOD_bkgnd_flag_t* flag, *TCOD_alignment_t* alignment, **const** char *fmt, ...)

Format and print a UTF-8 string to a console.

New in version 1.8.

int **TCOD_console_printf_rect**(TCOD_Console *con, int x, int y, int w, int h, **const** char *fmt, ...)

Format and print a UTF-8 string to a console.

New in version 1.8.

int **TCOD_console_printf_rect_ex**(TCOD_Console *con, int x, int y, int w, int h, *TCOD_bkgnd_flag_t* flag, *TCOD_alignment_t* alignment, **const** char *fmt, ...)

Format and print a UTF-8 string to a console.

New in version 1.8.

int **TCOD_console_get_height_rect_fmt**(**struct** TCOD_Console *con, int x, int y, int w, int h, **const** char *fmt, ...)

Return the number of lines that would be printed by this formatted string.

New in version 1.8.

void **TCOD_console_printf_frame**(**struct** TCOD_Console *con, int x, int y, int w, int h, int *empty*, *TCOD_bkgnd_flag_t* flag, **const** char *fmt, ...)

Print a framed and optionally titled region to a console, using default colors and alignment.

This function uses Unicode box-drawing characters and a UTF-8 formatted string. New in version 1.8.

2.2.6 Printing functions using wchar_t

Note: These functions say they are UTF, however they will behave as UCS2 or UCS4 depending on the platform.

void **TCOD_console_print_utf**(TCOD_Console *con, int x, int y, **const** wchar_t *fmt, ...)

Deprecated since version 1.8: Use *TCOD_console_printf()* instead.

void **TCOD_console_print_ex_utf**(TCOD_Console *con, int x, int y, *TCOD_bkgnd_flag_t flag*, *TCOD_alignment_t alignment*, **const** wchar_t *fmt, ...)

Deprecated since version 1.8: Use *TCOD_console_printf_ex()* instead.

int **TCOD_console_print_rect_utf**(TCOD_Console *con, int x, int y, int w, int h, **const** wchar_t *fmt, ...)

int **TCOD_console_print_rect_ex_utf**(TCOD_Console *con, int x, int y, int w, int h, *TCOD_bkgnd_flag_t flag*, *TCOD_alignment_t alignment*, **const** wchar_t *fmt, ...)

Deprecated since version 1.8: Use *TCOD_console_printf_rect_ex()* instead.

int **TCOD_console_get_height_rect_utf**(TCOD_Console *con, int x, int y, int w, int h, **const** wchar_t *fmt, ...)

Deprecated since version 1.8.

2.2.7 Reading the content of the console

int **TCOD_console_get_width**(**const** TCOD_Console *con)
Return the width of a console.

int **TCOD_console_get_height**(**const** TCOD_Console *con)
Return the height of a console.

int **TCOD_console_get_char**(**const** TCOD_Console *con, int x, int y)
Return a character code of a console at x,y.

Return The character code.

Parameters

- `con`: A console pointer.
- `x`: The X coordinate, the left-most position being 0.
- `y`: The Y coordinate, the top-most position being 0.

TCOD_color_t **TCOD_console_get_char_foreground**(**const** TCOD_Console *con, int x, int y)
Return the foreground color of a console at x,y.

Return A TCOD_color_t struct with a copy of the foreground color.

Parameters

- `con`: A console pointer.
- `x`: The X coordinate, the left-most position being 0.
- `y`: The Y coordinate, the top-most position being 0.

TCOD_color_t **TCOD_console_get_char_background**(**const** TCOD_Console *con, int *x*, int *y*)
Return the background color of a console at x,y.

Return A TCOD_color_t struct with a copy of the background color.

Parameters

- `con`: A console pointer.
- `x`: The X coordinate, the left-most position being 0.
- `y`: The Y coordinate, the top-most position being 0.

TCOD_color_t **TCOD_console_get_default_foreground**(TCOD_console_t *con*)

TCOD_color_t **TCOD_console_get_default_background**(TCOD_console_t *con*)

TCOD_bkgnd_flag_t **TCOD_console_get_background_flag**(TCOD_console_t *con*)
Return a consoles default background flag.

2.2.8 Screen fading functions

void **TCOD_console_set_fade**(uint8_t *val*, TCOD_color_t *fade*)
Fade the color of the display.

Parameters

- `val`: Where at 255 colors are normal and at 0 colors are completely faded.
- `fadecol`: Color to fade towards.

uint8_t **TCOD_console_get_fade**(void)
Return the fade value.

Return At 255 colors are normal and at 0 colors are completely faded.

TCOD_color_t **TCOD_console_get_fading_color**(void)
Return the fade color.

Return The current fading color.

2.2.9 ASCII constants

enum TCOD_chars_t
Values:

TCOD_CHAR_HLINE =196

TCOD_CHAR_VLINE =179

TCOD_CHAR_NE =191

TCOD_CHAR_NW =218

TCOD_CHAR_SE =217

TCOD_CHAR_SW =192

TCOD_CHAR_TEEW =180

TCOD_CHAR_TEEE =195

TCOD_CHAR_TEEN =193

TCOD_CHAR_TEES =194

TCOD_CHAR_CROSS =197

TCOD_CHAR_DHLINE =205

TCOD_CHAR_DVLINE =186

TCOD_CHAR_DNE =187

TCOD_CHAR_DNW =201

TCOD_CHAR_DSE =188

TCOD_CHAR_DSW =200

TCOD_CHAR_DTEEW =185

TCOD_CHAR_DTEEE =204

TCOD_CHAR_DTEEN =202

TCOD_CHAR_DTEES =203

TCOD_CHAR_DCROSS =206

TCOD_CHAR_BLOCK1 =176

TCOD_CHAR_BLOCK2 =177

TCOD_CHAR_BLOCK3 =178

TCOD_CHAR_ARROW_N =24

TCOD_CHAR_ARROW_S =25

TCOD_CHAR_ARROW_E =26

TCOD_CHAR_ARROW_W =27

TCOD_CHAR_ARROW2_N =30

TCOD_CHAR_ARROW2_S =31

TCOD_CHAR_ARROW2_E =16

TCOD_CHAR_ARROW2_W =17

TCOD_CHAR_DARROW_H =29

TCOD_CHAR_DARROW_V =18

TCOD_CHAR_CHECKBOX_UNSET =224

TCOD_CHAR_CHECKBOX_SET =225

TCOD_CHAR_RADIO_UNSET =9

TCOD_CHAR_RADIO_SET =10

TCOD_CHAR_SUBP_NW =226

TCOD_CHAR_SUBP_NE =227

TCOD_CHAR_SUBP_N =228

TCOD_CHAR_SUBP_SE =229

TCOD_CHAR_SUBP_DIAG =230

TCOD_CHAR_SUBP_E =231

TCOD_CHAR_SUBP_SW =232

TCOD_CHAR_SMILIE = 1

TCOD_CHAR_SMILIE_INV = 2

TCOD_CHAR_HEART = 3

TCOD_CHAR_DIAMOND = 4

TCOD_CHAR_CLUB = 5

TCOD_CHAR_SPADE = 6

TCOD_CHAR_BULLET = 7

TCOD_CHAR_BULLET_INV = 8

TCOD_CHAR_MALE = 11

TCOD_CHAR_FEMALE = 12

TCOD_CHAR_NOTE = 13

TCOD_CHAR_NOTE_DOUBLE = 14

TCOD_CHAR_LIGHT = 15

TCOD_CHAR_EXCLAM_DOUBLE = 19

TCOD_CHAR_PILCROW = 20

TCOD_CHAR_SECTION = 21

TCOD_CHAR_POUND = 156

TCOD_CHAR_MULTIPLICATION = 158

TCOD_CHAR_FUNCTION = 159

TCOD_CHAR_RESERVED = 169

TCOD_CHAR_HALF = 171

TCOD_CHAR_ONE_QUARTER = 172

TCOD_CHAR_COPYRIGHT = 184

TCOD_CHAR_CENT = 189

TCOD_CHAR_YEN = 190

TCOD_CHAR_CURRENCY = 207

TCOD_CHAR_THREE_QUARTERS = 243

TCOD_CHAR_DIVISION = 246

TCOD_CHAR_GRADE = 248

TCOD_CHAR_UMLAUT = 249

TCOD_CHAR_POW1 = 251

TCOD_CHAR_POW3 = 252

TCOD_CHAR_POW2 = 253

TCOD_CHAR_BULLET_SQUARE = 254

2.3 Flushing the root console

void **TCOD_console_flush**(void)
Render and present the root console to the active display.

2.4 Handling user input

2.4.1 Blocking user input

TCOD_key_t **TCOD_console_wait_for_keypress**(bool *flush*)
Wait for a key press event, then return it.

Do not solve input lag issues by arbitrarily dropping events!

Return A *TCOD_key_t* struct with the most recent key data.

Parameters

- `flush`: If 1 then the event queue will be cleared before waiting for the next event. This should always be 0.

TCOD_event_t **TCOD_sys_wait_for_event**(int *eventMask*, *TCOD_key_t* **key*, *TCOD_mouse_t* **mouse*, bool *flush*)
Wait for a specific type of event.

This function also returns when the SDL window is being closed.

Return A TCOD_event_t flag showing which event was actually processed.

Parameters

- `eventMask`: A bit-mask of TCOD_event_t flags.
- `key`: Optional pointer to a *TCOD_key_t* struct.
- `mouse`: Optional pointer to a *TCOD_mouse_t* struct.
- `flush`: This should always be false.

2.4.2 Non blocking user input

TCOD_key_t **TCOD_console_check_for_keypress**(int *flags*)
Return immediately with a recently pressed key.

Return A *TCOD_key_t* struct with a recently pressed key. If no event exists then the `vk` attribute will be `TCODK_NONE`

Parameters

- `flags`: A TCOD_event_t bit-field, for example: `TCOD_EVENT_KEY_PRESS`

bool **TCOD_console_is_key_pressed**(*TCOD_keycode_t key*)

TCOD_event_t **TCOD_sys_check_for_event**(int *eventMask*, *TCOD_key_t* **key*, *TCOD_mouse_t* **mouse*)

Check for a specific type of event.

Return A TCOD_event_t flag showing which event was actually processed.

Parameters

- `eventMask`: A bit-mask of TCOD_event_t flags.
- `key`: Optional pointer to a *TCOD_key_t* struct.
- `mouse`: Optional pointer to a *TCOD_mouse_t* struct.
- `flush`: This should always be false.

TCOD_mouse_t **TCOD_mouse_get_status**(void)

Return a copy of the current mouse state.

2.4.3 Keyboard event structure

enum TCOD_key_status_t

Values:

TCOD_KEY_PRESSED =1

TCOD_KEY_RELEASED =2

struct TCOD_key_t

2.4.4 Key codes

enum TCOD_keycode_t

Values:

TCODK_NONE

TCODK_ESCAPE

TCODK_BACKSPACE

TCODK_TAB

TCODK_ENTER

TCODK_SHIFT

TCODK_CONTROL

TCODK_ALT

TCODK_PAUSE

TCODK_CAPSLOCK

TCODK_PAGEUP

TCODK_PAGEDOWN

TCODK_END

TCODK_HOME

TCODK_UP

TCODK_LEFT

TCODK_RIGHT

TCODK_DOWN

TCODK_PRINTSCREEN

TCODK_INSERT

TCODK_DELETE

TCODK_LWIN

TCODK_RWIN

TCODK_APPS

TCODK_0

TCODK_1

TCODK_2

TCODK_3

TCODK_4

TCODK_5

TCODK_6

TCODK_7

TCODK_8

TCODK_9

TCODK_KP0

TCODK_KP1

TCODK_KP2

TCODK_KP3

TCODK_KP4

TCODK_KP5

TCODK_KP6

TCODK_KP7

TCODK_KP8

TCODK_KP9

TCODK_KPADD

TCODK_KPSUB

TCODK_KPDIV

TCODK_KPMUL

TCODK_KPDEC

TCODK_KPENTER

TCODK_F1

TCODK_F2

TCODK_F3

TCODK_F4

TCODK_F5

TCODK_F6

TCODK_F7

TCODK_F8

TCODK_F9

TCODK_F10

TCODK_F11

TCODK_F12

TCODK_NUMLOCK

TCODK_SCROLLLOCK

TCODK_SPACE

TCODK_CHAR

TCODK_TEXT

2.4.5 Mouse event structure

struct TCOD_mouse_t

2.5 Using off-screen consoles

2.5.1 Creating and deleting off-screen consoles

TCOD_console_t **TCOD_console_new** (int *w*, int *h*)
Return a new console with a specific number of columns and rows.

Return A pointer to the new console, or NULL on error.

Parameters

- w: Number of columns.
- h: Number of columns.

void **TCOD_console_delete** (TCOD_console_t *console*)
Delete a console.

If the console being deleted is the root console, then the display will be uninitialized.

Parameters

- con: A console pointer.

2.5.2 Creating an off-screen console from any .asc/.apf/.xp file

TCOD_console_t **TCOD_console_from_file**(**const** char *_filename_)

2.5.3 Loading an offscreen console from a .asc file

bool **TCOD_console_load_asc**(TCOD_console_t _con_, **const** char *_filename_)

2.5.4 Loading an offscreen console from a .apf file

bool **TCOD_console_load_apf**(TCOD_console_t _con_, **const** char *_filename_)

2.5.5 Saving a console to a .asc file

bool **TCOD_console_save_asc**(TCOD_console_t _con_, **const** char *_filename_)

2.5.6 Saving a console to a .apf file

bool **TCOD_console_save_apf**(TCOD_console_t _con_, **const** char *_filename_)

2.5.7 Working with REXPaint `.xp` files

REXPaint gives special treatment to tiles with a magic pink `{255, 0, 255}` background color. You can processes this effect manually or by setting *`TCOD_console_set_key_color()`* to `TCOD_fuchsia`.

`libtcodpy`.**console_from_xp**(*filename*)

TCOD_console_t **TCOD_console_from_xp**(**const** char *_filename_)

Return a new console loaded from a REXPaint `.xp` file.

Return A new TCOD_console_t object. New consoles will need to be deleted with a call to :any:`TCOD_console_delete`. Returns NULL on an error.

Parameters

- `filename`: A path to the REXPaint file.

`libtcodpy`.**console_load_xp**(*con*, *filename*)

bool `TCODConsole`::**loadXp**(**const** char *_filename_)

bool **TCOD_console_load_xp**(TCOD_Console *_con_, **const** char *_filename_)

`libtcodpy`.**console_save_xp**(*con*, *filename*, *compress_level=-1*)

bool `TCODConsole`::**saveXp**(**const** char *_filename_, int _compress_level_)

bool **TCOD_console_save_xp**(**const** TCOD_Console *_con_, **const** char *_filename_, int _compress_level_)

Save a console as a REXPaint `.xp` file.

The REXPaint format can support a 1:1 copy of a libtcod console.

Return `true` when the file is saved succesfully, or `false` when an issue is detected.

Parameters

- `con`: The console instance to save.
- `filename`: The filepath to save to.
- `compress_level`: A zlib compression level, from 0 to 9. 1=fast, 6=balanced, 9=slowest, 0=uncompressed.

`libtcodpy.`**`console_list_from_xp`**(*filename*)

TCOD_list_t **`TCOD_console_list_from_xp`**(**`const`** char **filename*)

Return a list of consoles from a REXPaint file.

This function can load a REXPaint file with variable layer shapes, which would cause issues for a function like TCOD_console_list_from_xp.

Return Returns a TCOD_list_t of TCOD_console_t objects. Or NULL on an error. You will need to delete this list and each console individually.

Parameters

- `filename`: A path to the REXPaint file.

`libtcodpy.`**`console_list_save_xp`**(*console_list*, *filename*, *compress_level*)

bool **`TCOD_console_list_save_xp`**(TCOD_list_t *console_list*, **`const`** char **filename*, int *compress_level*)

Save a list of consoles to a REXPaint file.

This function can save any number of layers with multiple different sizes.

Return true on success, false on a failure such as not being able to write to the path provided.

Parameters

- `console_list`: A TCOD_list_t of TCOD_console_t objects.
- `filename`: Path to save to.
- `compress_level`: zlib compression level.

The REXPaint tool only supports files with up to 9 layers where all layers are the same size.

2.5.8 Blitting a console on another one

void **`TCOD_console_blit`**(TCOD_console_t *src*, int *xSrc*, int *ySrc*, int *wSrc*, int *hSrc*, TCOD_console_t *dst*, int *xDst*, int *yDst*, float *foreground_alpha*, float *background_alpha*)

Blit from one console to another.

If the source console has a key color, this function will use it.

Parameters

- `srcCon`: Pointer to the source console.
- `xSrc`: The left region of the source console to blit from.
- `ySrc`: The top region of the source console to blit from.
- `wSrc`: The width of the region to blit from. If 0 then it will fill to the maximum width.
- `hSrc`: The height of the region to blit from. If 0 then it will fill to the maximum height.
- `dstCon`: Pointer to the destination console.
- `xDst`: The left corner to blit onto the destination console.
- `yDst`: The top corner to blit onto the destination console.

- `foreground_alpha`: Foreground blending alpha.
- `background_alpha`: Background blending alpha.

2.5.9 Define a blit-transparent color

void **`TCOD_console_set_key_color`** (TCOD_console_t *con*, TCOD_color_t *col*)

CHAPTER 3

System layer

3.1 High precision time functions

3.1.1 Limit the frames per second

3.1.2 Get the number of frames rendered during the last second

3.1.3 Get the duration of the last frame

3.1.4 Pause the program

3.1.5 Get global timer in milliseconds

3.1.6 Get global timer in seconds

3.2 Easy screenshots

3.3 Filesystem utilities

3.4 Draw custom graphics on top of the root console

3.5 Miscellaneous utilities

3.6 Clipboard integration

CHAPTER 4

All Purpose Container

CHAPTER 5

Compression Toolkit

CHAPTER 6

File Parser

CHAPTER 7

Image Toolkit

CHAPTER 8

Line Drawing Toolkit

8.1 Iterator-based line drawing

```
#include <libtcod.h>

void main() {
  TCOD_bresenham_data_t bresenham_data;
  int x=5, y=8;
  TCOD_line_init_mt(x, y, 13, 14, &bresenham_data);
  do {
    printf("%d %d\n", x, y);
  } while (!TCOD_line_step_mt(&x, &y, &bresenham_data));
}
```

struct TCOD_bresenham_data_t
A struct used for computing a bresenham line.

void **TCOD_line_init_mt**(int *xFrom*, int *yFrom*, int *xTo*, int *yTo*, *TCOD_bresenham_data_t* **data*)
Initialize a *TCOD_bresenham_data_t* struct.

After calling this function you use TCOD_line_step_mt to iterate over the individual points on the line.

Parameters

- `xFrom`: The starting x position.
- `yFrom`: The starting y position.
- `xTo`: The ending x position.
- `yTo`: The ending y position.
- `data`: Pointer to a *TCOD_bresenham_data_t* struct.

bool **TCOD_line_step_mt**(int **xCur*, int **yCur*, *TCOD_bresenham_data_t* **data*)
Get the next point on a line, returns true once the line has ended.

The starting point is excluded by this function. After the ending point is reached, the next call will return true.

Return true after the ending point has been reached.

Parameters

- `xCur`: An int pointer to fill with the next x position.
- `yCur`: An int pointer to fill with the next y position.
- `data`: Pointer to a initialized *TCOD_bresenham_data_t* struct.

8.2 Callback-based line drawing

```
#include <libtcod.h>

void main() {
  bool my_listener(int x,int y) {
    printf("%d %d\n", x, y);
    return true;
  }
  TCOD_line(5, 8, 13, 4, my_listener);
}
```

typedef bool (***TCOD_line_listener_t**) (int x, int y)
A callback to be passed to TCOD_line.

The points given to the callback include both the starting and ending positions.

Return As long as this callback returns true it will be called with the next x,y point on the line.

Parameters

- `x`:
- `y`:

bool **TCOD_line** (int *xFrom*, int *yFrom*, int *xTo*, int *yTo*, *TCOD_line_listener_t listener*)
Iterate over a line using a callback.

Changed in version 1.6.6: This function is now reentrant.

Return true if the line was completely exhausted by the callback.

Parameters

- `xo`: The origin x position.
- `yo`: The origin y position.
- `xd`: The destination x position.
- `yd`: The destination y position.
- `listener`: A TCOD_line_listener_t callback.

8.3 Deprecated functions

bool **TCOD_line_mt** (int *xFrom*, int *yFrom*, int *xTo*, int *yTo*, *TCOD_line_listener_t listener*, *TCOD_bresenham_data_t* **data*)
Iterate over a line using a callback.

Deprecated since version 1.6.6: The *data* parameter for this call is redundant, you should call *TCOD_line()* instead.

Return true if the line was completely exhausted by the callback.

Parameters

- `xo`: The origin x position.
- `yo`: The origin y position.
- `xd`: The destination x position.
- `yd`: The destination y position.
- `listener`: A TCOD_line_listener_t callback.
- `data`: Pointer to a *TCOD_bresenham_data_t* struct.

void **TCOD_line_init** (int *xFrom*, int *yFrom*, int *xTo*, int *yTo*)

Initialize a line using a global state.

Deprecated since version 1.6.6: This function is not reentrant and will fail if a new line is started before the last is finished processing.

Use *TCOD_line_init_mt()* instead.

Parameters

- `xFrom`: The starting x position.
- `yFrom`: The starting y position.
- `xTo`: The ending x position.
- `yTo`: The ending y position.

bool **TCOD_line_step** (int **xCur*, int **yCur*)

Get the next point in a line, returns true once the line has ended.

The starting point is excluded by this function. After the ending point is reached, the next call will return true.

Return true once the ending point has been reached.

Parameters

- `xCur`: An int pointer to fill with the next x position.
- `yCur`: An int pointer to fill with the next y position.

Deprecated since version 1.6.6: This function is not reentrant and will fail if a new line is started before the last is finished processing.

Use *TCOD_line_step_mt()* instead.

CHAPTER 9

Mouse Support

CHAPTER 10

Noise Generator

CHAPTER 11

Psuedorandom Number Generator

CHAPTER 12

BSP

CHAPTER 13

Field of View

CHAPTER 14

Heightmap Toolkit

CHAPTER 15

Name Generator

CHAPTER 16

Pathfinding

CHAPTER 17

Upgrading

17.1 1.5.x -> 1.6.x

The largest and most influential change to libtcod, between versions 1.5.2 and 1.6.0, was the move to replace SDL with SDL2. SDL2 made many extensive changes to concepts used in SDL. Only one of these changes, the separation of text and key events, required a change in the libtcod API requiring users to update their code in the process of updating the version of libtcod they use.

When a user presses a key, they may be pressing `SHIFT` and =. On some keyboards, depending on the user's language and location, this may show + on the screen. On other user's keyboards, who knows what it may show on screen. SDL2 changes the way "the text which is displayed on the user's screen" is sent in key events. This means that the key event for `SHIFT` and = will be what happens for presses of both + and = (for user's with applicable keyboards), and there will be a new text event that happens with the displayed +.

17.1.1 In libtcod 1.5.x

SDL would when sending key events, provide the unicode character for the key event, ready for use. This meant that if the user happened to be using a British keyboard (or any that are similarly laid out), and pressed `SHIFT` and =, the event would be for the character +.

Listing 1: C / C++

```
if (key->c == '+') {
    /* Handle any key that displays a plus. */
}
```

Listing 2: Python

```
if key.c == "+":
    pass # Handle any key that displays a plus.
```

17.1.2 In libtcod 1.6.x

With SDL2, the raw key-presses still occur, but they are fundamentally linked to the keyboard of the user. Now there will still be an event where it says `SHIFT` and = are pressed, but the event will always be for the unmodified character =. The unicode text arrives in a new kind of event, and getting it requires explicitly checking that the event is the new text event, and then looking for the value in the relevant `text` field for the language being used.

Listing 3: C / C++

```
if (key->vk == TCODK_TEXT)
    if (key.text[0] == '+') {
        ; /* Handle any key that displays a plus. */
    }
```

Listing 4: Python

```
if key.vk == libtcod.KEY_TEXT:
    if key.text == "+":
        pass # Handle any key that displays a plus.
```

17.1.3 Still confused?

Run your code from a terminal or DOS window and print out the event attributes/fields and look at what is going on. Have your code print out the modifiers, the keycode, the character, the text, and then run it and try pressing some keys. It will be much faster than posting "I don't understand" or "Can someone explain" somewhere and waiting for a response.

Index

L

T