libtcod Documentation

Release 1.12.3

Richard Tew

Jun 06, 2019

CORE

CHAPTER

ONE

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.

1.11 C++

class TCODColor

STANDARD COLORS red 255,0,0 flame 255,63,0 orange 255,127,0 amber 255,191,0 yellow 255,255,0, lime 191,255,0 chartreuse 127,255,0 green 0,255,0 sea 0,255,127 turquoise 0,255,191 cyan 0,255,255 sky 0,191,255 azure 0,127,255 blue 0,0,255 han 63,0,255 violet 127,0,255 purple 191,0,255 fuchsia 255,0,255 magenta 255,0,191 pink 255,0,127 crimson 255,0,63 METALLIC COLORS brass 191,151,96 copper 196,136,124 gold 229,191,0 silver 203,203,203 MISCELLANEOUS COLORS celadon 172,255,175 peach 255,159,127 GREYSCALE & SEPIA grey 127,127,127 sepia 127,101,63 BLACK AND WHITE black 0,0,0 white 255,255,255

color Core Colors The Doryen library uses 32bits colors. Thus, your OS desktop must use 32bits colors. 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): TCODColor::desaturatedRed TCODColor::lightestRed TCODColor::lighterRed TCODColor::lightRed TCODColor::red TCODColor::darkRed TCODColor::darkerRed TCODColor::darkestRed TCOD_desaturated_red TCOD_lightest_red TCOD_lighter_red TCOD_light_red TCOD_red TCOD_dark_red TCOD_darker_red TCOD_darkest_red libtcod.desaturated_red libtcod.lightest_red libtcod.lighter_red libtcod.light_red libtcod.red libtcod.dark_red libtcod.darker_red libtcod.darkest_red #Ex TCODColor::desaturatedRed TCODColor::lightestRed TCODColor::lighterRed TCODColor::lightRed TCODColor::red TCODColor::darkRed TCODColor::darkerRed TCODColor::darkestRed tcod.color.desaturatedRed tcod.color.lightestRed tcod.color.lighterRed tcod.color.lightRed tcod.color.red tcod.color.darkRed tcod.color.darkerRed tcod.color.darkestRed

Public Functions

TCODColor(uint8_t *r_*, uint8_t *g_*, uint8_t *b_*)

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

TCODColor myColor(24,64,255); //RGB *TCODColor* myOtherColor(321.0f,0.7f,1.0f); //HSV TCOD_color_t my_color={24,64,255}; /* RGB */ TCOD_color_t my_other_color = TCOD_color_RGB(24,64,255); /* RGB too */ TCOD_color_t my_yet_another_color = TCOD_color_HSV(321.0f,0.7f,1.0f); /* HSV */ my_color=libtcod.Color(24,64,255) #Ex *TCODColor* myColor = new *TCODColor(24,64,255)*; //RGB *TCODColor* myColor = new *TCODColor*(321.0f,0.7f,1.0f); //HSV myColor = tcod.Color(24,24,255)

bool **operator==** (**const** *TCODColor* &*c*) **const**

color Compare two colors if (myColor == TCODColor::yellow) { ...

} if (myColor != TCODColor::white) { ... } if (TCOD_color_equals(my_color,TCOD_yellow)) { ... } if (!TCOD_color_equals(my_color,TCOD_white)) { ... } if my_color == libtcod.yellow : ... if my_color != litbcod.white : ... #Ex if (myColor.Equal(TCODColor.yellow)) { ... } if (myColor.NotEqual(TCODColor.white)) { ... } if myColor == tcod.color.yellow then ... end

TCODColor **operator*** (**const** *TCODColor* &*rhs*) **const**

color Multiply two colors c1 = c2 * c3 => c1.r = c2.r * c3.r / 255 c1.g = c2.g * c3.g / 255 c1.b = c2.b * c3.b / 255 darkishRed = darkGrey * red

TCODColor myDarkishRed = TCODColor::darkGrey * TCODColor::lightRed; TCOD_color_t my_darkish_red = TCOD_color_multiply(TCOD_dark_grey, TCOD_light_red); my_darkish_red = libtcod.dark_grey * libtcod.light_red #Ex *TCODColor* myDarkishRed = TCODColor.darkGrey.Multiply(TCODColor.lightRed); myDarkishRed = tcod.color.darkGrey * tcod.color.lightRed

TCODColor **operator*** (float *value*) **const**

color Multiply a color by a float c1 = c2 * v => c1.r = CLAMP(0, 255, c2.r * v) c1.g = CLAMP(0, 255, c2.g * v) c1.b = CLAMP(0, 255, c2.b * v) darkishRed = red * 0.5

</tbody> *TCODColor* myDarkishRed = TCODColor::lightRed * 0.5f; TCOD_color_t my_darkish_red = TCOD_color_multiply_scalar(TCOD_light_red, 0.5f); myDarkishRed = litbcod.light_red * 0.5 #Ex *TCODColor* myDarkishRed = TCODColor.lightRed.Multiply(0.5f); myDarkishRed = tcod.color.lightRed * 0.5

TCODColor **operator+** (**const** *TCODColor* &*rhs*) **const**

color Adding two colors c1 = c1 + c2 => c1.r = MIN(255, c1.r + c2.r) c1.g = MIN(255, c1.g + c2.g) c1.b = MIN(255, c1.b + c2.b) lightishRed = red + darkGrey

TCODColor myLightishRed = TCODColor::red + TCODColor::darkGrey TCOD_color_t my_lightish_red = TCOD_color_add(TCOD_red, TCOD_dark_grey); myLightishRed = libtcod.red + libtcod.dark_grey #Ex *TCODColor* myLightishRed = TCODColor.red.Plus(TCODColor.darkGrey) myLightishRed = tcod.color.red + tcod.color.darkGrey

TCODColor **operator-** (**const** *TCODColor* &*rhs*) **const**

color Subtract two colors c1 = c1 - c2 => c1.r = MAX(0, c1.r - c2.r) c1.g = MAX(0, c1.g - c2.g) c1.b = MAX(0, c1.b - c2.b) redish = red - darkGrey

TCODColor myRedish = TCODColor::red - TCODColor::darkGrey TCOD_color_t my_redish = TCOD_color_subtract(TCOD_red, TCOD_dark_grey); myRedish = libtcod.red - libtcod.dark_grey #Ex *TCODColor* myRedish = TCODColor.red.Minus(TCODColor.darkGrey) myRedish = tcod.color.red - tcod.color.darkGrey

void **setHSV** (float *h*, float *s*, float *v*)

color 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 *TCODColor::setHSV(float h, float s, float v)* void TCOD_color_set_HSV(TCOD_color_t *c,float h, float s, float v) color_set_hsv(c,h,s,v) # void *TCODColor::setHSV(float h, float s, float v)* Color:setHSV(h, s ,v) c In the C and Python versions, the color to modify h,s,v Color components in the HSV space 0.0 <= h < 360.0 0.0 <= s <= 1.0 0.0 <= v <= 1.0

void **setHue** (float *h*)

color Define a color's hue, saturation or lightness These functions set only a single component in the HSV color space.

void *TCODColor::setHue* (float h) void TCODColor::setSaturation (float s) void TCODColor::setValue (float v) void TCOD_color_set_hue (TCOD_color_t *c, float h) void TCOD_color_set_saturation (TCOD_color_t *c, float s) void TCOD_color_set_value (TCOD_color_t *c, float v) Color:setHue(h) Color:setSaturation(s) Color:setValue(v) h,s,v Color components in the HSV space c In the C and Python versions, the color to modify

void **getHSV** (float **h*, float **s*, float **v*) **const**

color Get a color hue, saturation and value components void *TCODColor::getHSV(float *h, float *s, float *v) const* void TCOD_color_get_HSV(TCOD_color_t c,float * h, float * s, float * v) color_get_HSV(c) # returns [h,s,v] # void TCODColor::getHSV(out float h, out float s, out float v) Color:*getHSV()* returns h,s,v c In the C and Python versions, the TCOD_color_t from which to read.

h,s,v Color components in the HSV space 0.0 <= h < 360.0 0.0 <= s <= 1.0 0.0 <= v <= 1.0

float **getHue** ()

color Get a color's hue, saturation or value 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 *TCODColor::getHue* () float TCODColor::getSaturation () float TCODColor::getValue () float TCOD_color_get_hue (TCOD_color_t c) float TCOD_color_get_saturation (TCOD_color_t c) float TCOD_color_get_value (TCOD_color_t c) Color:*getHue()* Color:getSaturation() Color:getValue() # float *TCODColor::getHue()* float TCODColor::getSaturation() float TCODColor::getValue() c the TCOD_color_t from which to read

void **shiftHue** (float *hshift*)

color 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 *TCODColor::shiftHue* (float hshift) void TCOD_color_shift_hue (TCOD_color_t *c, float hshift) # *TCODColor::shiftHue(float hshift)* Color:shiftHue(hshift) c The color to modify hshift The hue shift value

void **scaleHSV** (float *sscale*, float *vscale*)

color Scale a color's saturation and value void *TCODColor::scaleHSV* (float sscale, float vscale) void TCOD_color_scale_HSV (TCOD_color_t *c, float scoef, float vcoef) color_scale_HSV(c, scoef, vcoef) # *TCODColor::scaleHSV* (float sscale, float vscale) Color:scaleHSV(sscale,vscale) c The color to modify sscale saturation multiplier (1.0f for no change) vscale value multiplier (1.0f for no change)

Public Static Functions

static *TCODColor* **lerp** (**const** *TCODColor* &*c1*, **const** *TCODColor* &*c2*, float *coef*)

color Interpolate between two colors c1 = lerp (c2, c3, coef) => c1.r = c2.r + (c3.r - c2.r) * coef c1.g = c2.g + (c3.g - c2.g) * coef c1.b = c2.b + (c3.b - c2.b) * coef coef should be between 0.0 and 1.0 but you can as well use other values

TCODColor myColor = *TCODColor::lerp* (TCODColor::darkGrey, TCODColor::lightRed,coef); TCOD_color_t my_color = TCOD_color_lerp (TCOD_dark_grey, TCOD_light_red,coef); my_color = libtcod.color_lerp (libtcod.dark_grey, litbcod.light_red,coef) #Ex *TCODColor* myColor = TCODColor.Interpolate(TCODColor.darkGrey, TCODColor.lightRed, coef); myColor = tcod.color.Interpolate(tcod.color.darkGrey, tcod.color.lightRed, coef)

void **genMap** (*TCODColor* **map*, int *nbKey*, *TCODColor* **const** **keyColor*, int **const** **keyIndex*)

color 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 :

black

red

white </tbody>

static void *genMap(TCODColor *map, int nbKey, TCODColor const *keyColor, int const *keyIndex)* void TCOD_gen_map(TCOD_color_t *map, int nb_key, TCOD_color_t const *key_color, int const *key_index) color_gen_map(keyColor,keyIndex) # returns an array of colors map An array of colors to be filled by the function. nbKey Number of color keys keyColor Array of nbKey colors containing the color of each key keyIndex Array of nbKey integers containing the index of each key. If you want to fill the map array, keyIndex[0] must be 0 and keyIndex[nbKey-1] is the number of elements in map minus 1 but you can also use the function to fill only a part of the map array. int idx[] = { 0, 4, 8 }; // indexes of the keys *TCODColor* col[] = { *TCODColor(0,0,0)*, *TCODColor(255,0,0)*, *TCODColor(255,255,255)* }; // colors : black, red, white *TCODColor* map[9]; TCODColor::genMap(map,3,col,idx); int idx[] = { 0, 4, 8 }; // indexes of the keys TCOD_color_t col[] = { { 0,0,0 }, {255,0,0}, {255,255,255} }; // colors : black, red, white TCOD_color_t map[9]; TCOD_color_gen_map(map,3,col,idx); idx = [0, 4, 8] # indexes of the keys col = [libtcod.Color(0,0,0), libtcod.Color(255,0,0), libtcod.Color(255,255,255)] # colors : black, red, white map=libtcod.color_gen_map(col,idx)

CHAPTER

TWO

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 2.0 core renderer.

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

Changed in version 1.11.0: This renderer now uses OpenGL 2.0 instead of 2.1.

TCOD_NB_RENDERERS

int **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.

Returns 0 on success, or -1 on an error, you can check the error with TCOD_sys_get_error() Changed in version 1.12: Now returns -1 on error instead of crashing.

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.

TCOD_FONT_LAYOUT_CP437 = 16
Decode a code page 437 tileset into Unicode code-points.

New in version 1.10.

int **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.

Returns 0 on success, or -1 on an error, you can check the error with TCOD_sys_get_error() Changed in version 1.12: Now returns -1 on error instead of crashing.

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 **con*, TCOD_color_t *col*)

void **`TCOD_console_set_default_background`**(TCOD_Console **con*, TCOD_color_t *col*)

void **`TCOD_console_set_background_flag`**(TCOD_Console **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 **con*)
Clear a console to its default colors and the space character code.

void **`TCOD_console_put_char`**(TCOD_Console **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 **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 **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 **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 **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 **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 **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 **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`

Background color blend modes.

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 **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 **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 **con*)

TCOD_color_t **TCOD_console_get_default_background**(TCOD_Console **con*)

TCOD_bkgnd_flag_t **TCOD_console_get_background_flag**(TCOD_Console **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.

int **TCOD_sys_accumulate_console** (**const** TCOD_Console **console*)

Render a console over the display.

console can be any size, the active render will try to scale it to fit the screen.

The function will only work for the SDL2/OPENGL2 renderers.

Unlike `TCOD_console_flush()` this will not present the display. You will need to do that manually, likely with the SDL API.

Returns 0 on success, or a negative number on a failure such as the incorrect renderer being active.

New in version 1.11.

See also:

`TCOD_sys_get_sdl_window()` `TCOD_sys_get_sdl_renderer()`

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

2.4.6 Events from SDL2

TCOD_event_t `tcod::sdl2::`**`process_event`**(**const union** SDL_Event &*in*, *TCOD_key_t* &*out*)

Parse an SDL_Event into a key event and return the relevant TCOD_event_t.

Returns TCOD_EVENT_NONE if the event wasn't keyboard related. New in version 1.11.

TCOD_event_t **`TCOD_sys_process_key_event`**(**const union** SDL_Event **in*, *TCOD_key_t* **out*)

Parse an SDL_Event into a key event and return the relevant TCOD_event_t.

Returns TCOD_EVENT_NONE if the event wasn't keyboard related. New in version 1.11.

TCOD_event_t `tcod::sdl2::`**`process_event`**(**const union** SDL_Event &*in*, *TCOD_mouse_t* &*out*)

Parse an SDL_Event into a mouse event and return the relevant TCOD_event_t.

Returns TCOD_EVENT_NONE if the event wasn't mouse related. New in version 1.11.

TCOD_event_t **`TCOD_sys_process_mouse_event`**(**const union** SDL_Event **in*, *TCOD_mouse_t* **out*)

Parse an SDL_Event into a mouse event and return the relevant TCOD_event_t.

Returns TCOD_EVENT_NONE if the event wasn't mouse related. New in version 1.11.

2.5 Using off-screen consoles

2.5.1 Creating and deleting off-screen consoles

TCOD_Console ***`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 **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

- `[in] 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

- `[in] con`: The console instance to save.
- `[in] filename`: The filepath to save to.
- `[in] 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

- `[in] 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

- `[in] console_list`: A TCOD_list_t of TCOD_console_t objects.
- `[in] filename`: Path to save to.
- `[in] 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`**(**const** TCOD_Console **src*, int *xSrc*, int *ySrc*, int *wSrc*, int *hSrc*, TCOD_Console **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 **con*, TCOD_color_t *col*)

2.6 C++

`class TCODConsole`

console Core Console The console emulator handles the rendering of the game screen and the keyboard input.

Classic real time game loop: *TCODConsole::initRoot*(80,50,"my game",false); TCODSystem::setFps(25); // limit framerate to 25 frames per second while (!endGame && !TCODConsole::isWindowClosed()) {

TCOD_key_t key; TCODSystem::checkForEvent(TCOD_EVENT_KEY_PRESS,&key,NULL); updateWorld (key, *TCODSystem::getLastFrameLength()*); updateWorld(TCOD_key_t key, float elapsed) (using key if key.vk != TCODK_NONE) use elapsed to scale any update that is time dependent. . . . draw world+GUI on *TCODConsole::root* *TCODConsole::flush()*; } tcod.console.initRoot(80,50,"my game", false) root=libtcod.TCODConsole_root tcod.system.setFps(25) while not tcod.console.isWindowClosed() do . . . draw on root tcod.console.flush() key=tcod.console.checkForKeypress() . . . update world, using key and tcod.system.getLastFrameLength end console Classic turn by turn game loop: *TCODConsole::initRoot*(80,50,"my game",false); while (!endGame && !TCODConsole::isWindowClosed()) { . . . draw on *TCODConsole::root* *TCODConsole::flush()*; *TCOD_key_t* key; TCODConsole::waitForEvent(TCOD_EVENT_KEY_PRESS,&key,NULL,true); . . . update world, using key }

Public Functions

void **setDefaultBackground**(*TCODColor back*)

console_draw Drawing on the root console console

console_draw_basic Basic printing functions console_draw Setting the default background color This function changes the default background color for a console. The default background color is used by several drawing functions like clear, putChar, . . . void *TCODConsole::setDefaultBackground(TCODColor back)* void TCOD_console_set_default_background(TCOD_console_t con,TCOD_color_t back) console_set_default_background(con,back) # void TCODConsole::setBackgroundColor(TCODColor back) Console:setBackgroundColor(back) con in the C and Python versions, the offscreen console handler or NULL for the root console back the new default background color for this console *TCODConsole::root*->setDefaultBackground(myColor) TCOD_console_set_default_background(NULL, my_color) litbcod.console_set_default_background(0, my_color) libtcod.TCODConsole_root:setBackgroundColor(myColor)

void **setDefaultForeground**(*TCODColor fore*)

console_draw_basic Setting the default foreground color This function changes the default foreground color for a console.

The default foreground color is used by several drawing functions like clear, putChar, . . . void *TCODConsole::setDefaultForeground(TCODColor fore)* void TCOD_console_set_default_foreground(TCOD_console_t con,TCOD_color_t fore) console_set_default_foreground(con, fore) # void TCODConsole::setForegroundColor(TCODColor fore) Console:setForegroundColor(fore) con in the C and Python versions, the offscreen console handler or NULL for the root console fore the new default foreground color for this console *TCODConsole::root*->setDefaultForeground(myColor) TCOD_console_set_default_foreground(NULL, my_color) litbcod.console_set_default_foreground(0, my_color) libtcod.TCODConsole_root:setForegroundColor(myColor)

void **clear**()

console_draw_basic Clearing a console This function modifies all cells of a console : set the cell's background color to the console default background color set the cell's foreground color to the console default foreground color set the cell's ASCII code to 32 (space) void *TCODConsole::clear()* void TCOD_console_clear(TCOD_console_t con) console_clear(con) # void *TCODConsole::clear()* Console:*clear()* con in the C and Python versions, the offscreen console handler or NULL for the root console

void **setCharBackground**(int *x*, int *y*, **const** *TCODColor &col*, *TCOD_bkgnd_flag_t flag = TCOD_BKGND_SET*)

console_draw_basic Setting the background color of a cell This function modifies the background color of a cell, leaving other properties (foreground color and ASCII code) unchanged.

void *TCODConsole::setCharBackground*(int x, int y, const *TCODColor* &col, TCOD_bkgnd_flag_t flag = TCOD_BKGND_SET) void TCOD_console_set_char_background(TCOD_console_t con,int x, int y, TCOD_color_t col, TCOD_bkgnd_flag_t flag) console_set_char_background(con, x, y, col, flag=BKGND_SET) # void TCODConsole::setCharBackground(int x, int y, TCODColor col) void TCODConsole::setCharBackground(int x, int y, TCODColor col, TCODBackgroundFlag flag) Console:setCharBackground(x, y, col) Console:setCharBackground(x, y, col, flag) con in the C and Python versions, the offscreen console handler or NULL for the root console x,y coordinates of the cell in the console. 0 <= x < console width 0 <= y < console height col the background color to use. You can use color constants flag this flag defines how the cell's background color is modified. See TCOD_bkgnd_flag_t

void **setCharForeground** (int *x*, int *y*, **const** *TCODColor* &*col*)

console_draw_basic Setting the foreground color of a cell This function modifies the foreground color of a cell, leaving other properties (background color and ASCII code) unchanged.

void *TCODConsole::setCharForeground(int x, int y, const TCODColor &col)* void TCOD_console_set_char_foreground(TCOD_console_t con,int x, int y, TCOD_color_t col) console_set_char_foreground(con, x, y, col) # void TCODConsole::setCharForeground(int x, int y, TCODColor col) Console:setCharForeground(x, y, col) con in the C and Python versions, the offscreen console handler or NULL for the root console x,y coordinates of the cell in the console. 0 <= x < console width 0 <= y < console height col the foreground color to use. You can use color constants

void **setChar** (int *x*, int *y*, int *c*)

console_draw_basic Setting the ASCII code of a cell This function modifies the ASCII code of a cell, leaving other properties (background and foreground colors) unchanged.

Note that since a clear console has both background and foreground colors set to black for every cell, using setchar will produce black characters on black background. Use putchar instead. void *TCODConsole::setChar(int x, int y, int c)* void TCOD_console_set_char(TCOD_console_t con,int x, int y, int c) console_set_char(con, x, y, c) # void *TCODConsole::setChar(int x, int y, int c)* Console:setChar(x, y, c) con in the C and Python versions, the offscreen console handler or NULL for the root console x,y coordinates of the cell in the console. 0 <= x < console width 0 <= y < console height c the new ASCII code for the cell. You can use ASCII constants

void **putChar** (int *x*, int *y*, int *c*, *TCOD_bkgnd_flag_t flag* = *TCOD_BKGND_DEFAULT*)

console_draw_basic Setting every property of a cell using default colors This function modifies every property of a cell : update the cell's background color according to the console default background color (see TCOD_bkgnd_flag_t).

set the cell's foreground color to the console default foreground color set the cell's ASCII code to c void *TCODConsole::putChar*(int x, int y, int c, TCOD_bkgnd_flag_t flag = TCOD_BKGND_DEFAULT) void TCOD_console_put_char(TCOD_console_t con,int x, int y, int c, TCOD_bkgnd_flag_t flag) console_put_char(con, x, y, c, flag=BKGND_DEFAULT) # void TCODConsole::putChar(int x, int y, int c) void TCODConsole::putChar(int x, int y, int c, TCODBackgroundFlag flag) Console:putChar(x, y, c) Console:putChar(x, y, c, flag) con in the C and Python versions, the offscreen console handler or NULL for the root console x,y coordinates of the cell in the console. 0 <= x < console width 0 <= y < console height c the new ASCII code for the cell. You can use ASCII constants flag this flag defines how the cell's background color is modified. See TCOD_bkgnd_flag_t

void **putCharEx** (int *x*, int *y*, int *c*, **const** *TCODColor* &*fore*, **const** *TCODColor* &*back*)

console_draw_basic Setting every property of a cell using specific colors This function modifies every property of a cell : set the cell's background color to back.

set the cell's foreground color to fore. set the cell's ASCII code to c. void *TCODConsole::putCharEx(int x, int y, int c, const TCODColor & fore, const TCODColor & back)* void TCOD_console_put_char_ex(TCOD_console_t con,int x, int y, int c, TCOD_color_t fore, TCOD_color_t back) console_put_char_ex(con, x, y, c, fore, back) # void TCODConsole::putCharEx(int x, int y, int c, TCODColor fore, TCODColor back) Console:putCharEx(x, y, c, fore, back) con in the C and Python versions, the offscreen console handler or NULL for the root console x,y coordinates of the cell in the console. 0 <= x < console width 0 <= y < console height c the new ASCII code for the cell. You can use ASCII constants fore,back new foreground and background colors for this cell void **setBackgroundFlag**(*TCOD_bkgnd_flag_t flag*)

console_bkgnd_flag_t Background effect flags console_draw This flag is used by most functions that modify a cell background color.

It defines how the console's current background color is used to modify the cell's existing background color : TCOD_BKGND_NONE : the cell's background color is not modified. TCOD_BKGND_SET : the cell's background color is replaced by the console's default background color : newbk = curbk. TCOD_BKGND_MULTIPLY : the cell's background color is multiplied by the console's default background color : newbk = oldbk * curbk TCOD_BKGND_LIGHTEN : newbk = MAX(oldbk,curbk) TCOD_BKGND_DARKEN : newbk = MIN(oldbk,curbk) TCOD_BKGND_SCREEN : newbk = white - (white - oldbk) * (white - curbk) // inverse of multiply : (1-newbk) = (1-oldbk)*(1-curbk) TCOD_BKGND_COLOR_DODGE : newbk = curbk / (white - oldbk) TCOD_BKGND_COLOR_BURN : newbk = white - (white - oldbk) / curbk TCOD_BKGND_ADD : newbk = oldbk + curbk TCOD_BKGND_ADDALPHA(alpha) : newbk = oldbk + alpha*curbk TCOD_BKGND_BURN : newbk = oldbk + curbk - white TCOD_BKGND_OVERLAY : newbk = curbk.x <= 0.5 ? 2*curbk*oldbk : white - 2*(white-curbk)*(white-oldbk) TCOD_BKGND_ALPHA(alpha) : newbk = (1.0f-alpha)*oldbk + alpha*(curbk-oldbk) TCOD_BKGND_DEFAULT : use the console's default background flag Note that TCOD_BKGND_ALPHA and TCOD_BKGND_ADDALPHA are MACROS that needs a float parameter between (0.0 and 1.0). TCOD_BKGND_ALPH and TCOD_BKGND_ADDA should not be used directly (else they will have the same effect as TCOD_BKGND_NONE). For Python, remove TCOD_ : libtcod.BKGND_NONE For C# : None, Set, Multiply, Lighten, Darken, Screen, ColodDodge, ColorBurn, Add, Burn Overlay, Default With lua, use tcod.None, ..., tcod.Default, BUT tcod.console.Alpha(value) and tcod.console.AddAlpha(value) console_print String drawing functions console_draw Setting the default background flag This function defines the background mode (see TCOD_bkgnd_flag_t) for the console. This default mode is used by several functions (print, printRect, ...) void *TCODConsole::setBackgroundFlag(TCOD_bkgnd_flag_t flag)* void TCOD_console_set_background_flag(TCOD_console_t con,TCOD_bkgnd_flag_t flag) console_set_background_flag(con, flag) # void TCODConsole::setBackgroundFlag(TCODBackgroundFlag flag) Console:setBackgroundFlag(flag) con in the C and Python versions, the offscreen console handler or NULL for the root console flag this flag defines how the cell's background color is modified. See TCOD_bkgnd_flag_t

TCOD_bkgnd_flag_t **getBackgroundFlag**() **const**

console_print Getting the default background flag This function returns the background mode (see TCOD_bkgnd_flag_t) for the console.

This default mode is used by several functions (print, printRect, ...) TCOD_bkgnd_flag_t *TCODConsole::getBackgroundFlag() const* TCOD_bkgnd_flag_t TCOD_console_get_background_flag(TCOD_console_t con) console_get_background_flag(con) # TCODBackgroundFlag *TCODConsole::getBackgroundFlag()* Console:*getBackgroundFlag()* con in the C and Python versions, the offscreen console handler or NULL for the root console

void **setAlignment**(*TCOD_alignment_t alignment*)

console_print Setting the default alignment This function defines the default alignment (see TCOD_alignment_t) for the console.

This default alignment is used by several functions (print, printRect, ...). Values for alignment : TCOD_LEFT, TCOD_CENTER, TCOD_RIGHT (in Python, remove TCOD_ : libtcod.LEFT). For C# and Lua : LeftAlignment, RightAlignment, CenterAlignment void *TCODConsole::setAlignment(TCOD_alignment_t alignment)* void TCOD_console_set_alignment(TCOD_console_t con,TCOD_bkgnd_flag_t alignment) console_set_alignment(con, alignment) # void TCODConsole::setAlignment(TCODAlignment alignment) Console:setAlignment(alignment) con in the C and Python versions, the offscreen console handler or NULL for the root console alignment defines how the strings are printed on screen.

TCOD_alignment_t **getAlignment**() **const** console_print Getting the default alignment This function returns the default alignment (see TCOD_alignment_t) for the console.

This default mode is used by several functions (print, printRect, . . .). Values for alignment : TCOD_LEFT, TCOD_CENTER, TCOD_RIGHT (in Python, remove TCOD_ : libtcod.LEFT). For C# and Lua : LeftAlignment, RightAlignment, CenterAlignment TCOD_alignment_t *TCODConsole::getAlignment() const* TCOD_alignment_t TCOD_console_get_alignment(TCOD_console_t con) console_get_alignment(con) # TCODAlignment *TCODConsole::getAlignment()* Console:*getAlignment()* con in the C and Python versions, the offscreen console handler or NULL for the root console

void **print** (int *x*, int *y*, **const** char **fmt*, ...)
Print an EASCII formatted string to the console.

Deprecated EASCII function.

Deprecated since version 1.8: EASCII is being phased out. Use TCODConsole::printf or one of the UTF-8 overloads.

void **print** (int *x*, int *y*, **const** std::string &*str*)
Print a UTF-8 string to the console.

This method will use this consoles default alignment, blend mode, and colors. New in version 1.8.

void **print** (int *x*, int *y*, **const** std::string &*str*, *TCOD_alignment_t alignment*, *TCOD_bkgnd_flag_t flag*)
Print a UTF-8 string to the console with specific alignment and blend mode.

New in version 1.8.

void **printf** (int *x*, int *y*, **const** char **fmt*, ...)
Format and print a UTF-8 string to the console.

This method will use this consoles default alignment, blend mode, and colors. New in version 1.8.

void **printf** (int *x*, int *y*, *TCOD_bkgnd_flag_t flag*, *TCOD_alignment_t alignment*, **const** char **fmt*, ...)
Format and print a UTF-8 string to the console with specific alignment and blend mode.

New in version 1.8.

void **printEx** (int *x*, int *y*, *TCOD_bkgnd_flag_t flag*, *TCOD_alignment_t alignment*, **const** char **fmt*, ...)
Print an EASCII formatted string to the console.

Deprecated EASCII function.

Deprecated since version 1.8: Use *TCODConsole::print* or *TCODConsole::printf*. These functions have overloads for specifying flag and alignment.

int **printRect** (int *x*, int *y*, int *w*, int *h*, **const** char **fmt*, ...)
console_print Printing a string with default parameters and autowrap This function draws a string in a rectangle inside the console, using default colors, alignment and background mode.

If the string reaches the borders of the rectangle, carriage returns are inserted. If h > 0 and the bottom of the rectangle is reached, the string is truncated. If h = 0, the string is only truncated if it reaches the bottom of the console. The function returns the height (number of console lines) of the printed string. int *TCODConsole::printRect*(int x, int y, int w, int h, const char *fmt, . . .) int TCOD_console_print_rect(TCOD_console_t con,int x, int y, int w, int h, const char *fmt, . . .) console_print_rect(con, x, y, w, h, fmt) # int TCODConsole::printRect(int x, int y, int w, int h, string fmt) Console:printRect(x, y, w, h, fmt) con in the C and Python versions, the offscreen console handler or NULL for the root console x,y coordinate of the character in the console, depending on the alignment : TCOD_LEFT

: leftmost character of the string TCOD_CENTER : center character of the string TCOD_RIGHT : rightmost character of the string w,h size of the rectangle x <= x+w < console width y <= y+h < console height fmt printf-like format string, eventually followed by parameters. You can use control codes to change the colors inside the string, except in C#.

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

console_print Printing a string with specific alignment and background mode and autowrap This function draws a string in a rectangle inside the console, using default colors, but specific alignment and background mode.

If the string reaches the borders of the rectangle, carriage returns are inserted. If h > 0 and the bottom of the rectangle is reached, the string is truncated. If h = 0, the string is only truncated if it reaches the bottom of the console. The function returns the height (number of console lines) of the printed string. int *TCODConsole::printRectEx*(int x, int y, int w, int h, TCOD_bkgnd_flag_t flag, TCOD_alignment_t alignment, const char *fmt, …) int TCOD_console_print_rect_ex(TCOD_console_t con,int x, int y, int w, int h, TCOD_bkgnd_flag_t flag, TCOD_alignment_t alignment, const char *fmt, …) console_print_rect_ex(con, x, y, w, h, flag, alignment, fmt) # int TCODConsole::printRectEx(int x, int y, int w, int h, TCODBackgroundFlag flag, TCODAlignment alignment, string fmt) Console:printRectEx(x, y, w, h, flag, alignment, fmt) con in the C and Python versions, the offscreen console handler or NULL for the root console x,y coordinate of the character in the console, depending on the alignment : TCOD_LEFT : leftmost character of the string TCOD_CENTER : center character of the string TCOD_RIGHT : rightmost character of the string w,h size of the rectangle x <= x+w < console width y <= y+h < console height flag this flag defines how the cell's background color is modified. See TCOD_bkgnd_flag_t alignment defines how the strings are printed on screen. fmt printf-like format string, eventually followed by parameters. You can use control codes to change the colors inside the string, except in C#.

int **getHeightRect** (int *x*, int *y*, int *w*, int *h*, **const** char **fmt*, ...)

console_print Compute the height of an autowrapped string This function returns the expected height of an autowrapped string without actually printing the string with printRect or printRectEx int *TCODConsole::getHeightRect*(int x, int y, int w, int h, const char *fmt, …)

int TCOD_console_get_height_rect(TCOD_console_t con,int x, int y, int w, int h, const char *fmt, …) console_get_height_rect(con, x, y, w, h, fmt) # int TCODConsole::getHeightRect(int x, int y, int w, int h, string fmt) Console:getHeightRect(x, y, w, h, fmt) con in the C and Python versions, the offscreen console handler or NULL for the root console x,y coordinate of the rectangle upper-left corner in the console w,h size of the rectangle x <= x+w < console width y <= y+h < console height fmt printf-like format string, eventually followed by parameters. You can use control codes to change the colors inside the string, except in C#.

void **print** (int *x*, int *y*, **const** wchar_t **fmt*, ...)

console_print void *TCODConsole::print*(int x, int y, const wchar_t *fmt, …) void TCOD_console_print_utf(TCOD_console_t con,int x, int y, const wchar_t *fmt, …)

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

console_print void *TCODConsole::printEx*(int x, int y, TCOD_bkgnd_flag_t flag, TCOD_alignment_t alignment, const wchar_t *fmt, …) void TCOD_console_print_ex_utf(TCOD_console_t con,int x, int y, TCOD_bkgnd_flag_t flag, TCOD_alignment_t alignment, const wchar_t *fmt, …)

int **printRect** (int *x*, int *y*, int *w*, int *h*, **const** wchar_t **fmt*, ...)

console_print int *TCODConsole::printRect*(int x, int y, int w, int h, const wchar_t *fmt, …) int TCOD_console_print_rect_utf(TCOD_console_t con,int x, int y, int w, int h, const wchar_t *fmt, …)

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

console_print int *TCODConsole::printRectEx*(int x, int y, int w, int h, TCOD_bkgnd_flag_t flag, TCOD_alignment_t alignment, const wchar_t *fmt, ...) int TCOD_console_print_rect_ex_utf(TCOD_console_t con,int x, int y, int w, int h, TCOD_bkgnd_flag_t flag, TCOD_alignment_t alignment, const wchar_t *fmt, ...)

int **getHeightRect** (int *x*, int *y*, int *w*, int *h*, **const** wchar_t **fmt*, ...)

console_print int *TCODConsole::getHeightRect*(int x, int y, int w, int h, const wchar_t *fmt, ...) int TCOD_console_get_height_rect_utf(TCOD_console_t con,int x, int y, int w, int h, const wchar_t *fmt, ...)

void **rect** (int *x*, int *y*, int *w*, int *h*, bool *clear*, *TCOD_bkgnd_flag_t flag* = *TCOD_BKGND_DEFAULT*)

console_advanced console_draw Advanced printing functions Filling a rectangle with the background color Fill a rectangle inside a console.

For each cell in the rectangle : set the cell's background color to the console default background color if clear is true, set the cell's ASCII code to 32 (space) void *TCODConsole::rect*(int x, int y, int w, int h, bool clear, TCOD_bkgnd_flag_t flag = TCOD_BKGND_DEFAULT) void TCOD_console_rect(TCOD_console_t con,int x, int y, int w, int h, bool clear, TCOD_bkgnd_flag_t flag) console_rect(con,x, y, w, h, clear, flag=BKGND_DEFAULT) # void TCODConsole::rect(int x, int y, int w, int h, bool clear) void TCODConsole::rect(int x, int y, int w, int h, bool clear, TCODBackgroundFlag flag) Console:rect(x, y, w, h, clear) Console:rect(x, y, w, h, clear, flag) con in the C and Python versions, the offscreen console handler or NULL for the root console x,y coordinates of rectangle upper-left corner in the console. 0 <= x < console width 0 <= y < console height w,h size of the rectangle in the console. x <= x+w < console width y <= y+h < console height clear if true, all characters inside the rectangle are set to ASCII code 32 (space). If false, only the background color is modified flag this flag defines how the cell's background color is modified. See TCOD_bkgnd_flag_t

void **hline** (int *x*, int *y*, int *l*, *TCOD_bkgnd_flag_t flag* = *TCOD_BKGND_DEFAULT*)

console_advanced Drawing an horizontal line Draws an horizontal line in the console, using ASCII code TCOD_CHAR_HLINE (196), and the console's default background/foreground colors.

void *TCODConsole::hline*(int x,int y, int l, TCOD_bkgnd_flag_t flag = TCOD_BKGND_DEFAULT) void TCOD_console_hline(TCOD_console_t con,int x,int y, int l, TCOD_bkgnd_flag_t flag) console_hline(con,x,y,l,flag=BKGND_DEFAULT) # void TCODConsole::hline(int x,int y, int l) void TCODConsole::hline(int x,int y, int l, TCODBackgroundFlag flag) Console:hline(x,y, l) Console:hline(x,y, l, flag) con in the C and Python versions, the offscreen console handler or NULL for the root console x,y Coordinates of the line's left end in the console. 0 <= x < console width 0 <= y < console height l The length of the line in cells 1 <= l <= console width - x flag this flag defines how the cell's background color is modified. See TCOD_bkgnd_flag_t

void **vline** (int *x*, int *y*, int *l*, *TCOD_bkgnd_flag_t flag* = *TCOD_BKGND_DEFAULT*)

console_advanced Drawing an vertical line Draws an vertical line in the console, using ASCII code TCOD_CHAR_VLINE (179), and the console's default background/foreground colors.

void *TCODConsole::vline*(int x,int y, int l, TCOD_bkgnd_flag_t flag = TCOD_BKGND_DEFAULT) void TCOD_console_vline(TCOD_console_t con,int x,int y, int l, TCOD_bkgnd_flag_t flag) console_vline(con,x,y,l,flag=BKGND_DEFAULT) # void TCODConsole::vline(int x,int y, int l) void TCODConsole::vline(int x,int y, int l, TCODBackgroundFlag flag) Console:vline(x,y, l) Console:vline(x,y, l, flag) con in the C and Python versions, the offscreen console handler or NULL for the root console x,y Coordinates of the line's upper end in the console. 0 <= x < console width 0 <= y < console height l The length of the line in cells 1 <= l <= console height - y flag this flag defines how the cell's background color is modified. See TCOD_bkgnd_flag_t

void **printFrame** (int *x*, int *y*, int *w*, int *h*, bool *clear* = true, *TCOD_bkgnd_flag_t flag* = *TCOD_BKGND_DEFAULT*, **const** char **fmt* = NULL, ...)

console_advanced Drawing a window frame This function calls the rect function using the supplied background mode flag, then draws a rectangle with the console's default foreground color.

If fmt is not NULL, it is printed on the top of the rectangle, using inverted colors. void *TCODConsole::printFrame*(int x,int y,int w,int h, bool clear=true, TCOD_bkgnd_flag_t flag = TCOD_BKGND_DEFAULT, const char *fmt=NULL, ...) void TCOD_console_print_frame(TCOD_console_t con,int x,int y,int w,int h, bool clear, TCOD_bkgnd_flag_t flag, const char *fmt, ...) console_print_frame(con,x, y, w, h, clear=True, flag=BKGND_DEFAULT, fmt=0) # void TCODConsole::printFrame(int x,int y, int w,int h) void TCODConsole::printFrame(int x,int y, int w,int h, bool clear) void TCODConsole::printFrame(int x,int y, int w,int h, bool clear, TCODBackgroundFlag flag) void TCODConsole::printFrame(int x,int y, int w,int h, bool clear, TCODBackgroundFlag flag, string fmt) Console:printFrame(x,y, w,h) Console:printFrame(x,y, w,h, clear) Console:printFrame(x,y, w,h, clear, flag) Console:printFrame(x,y, w,h, clear, flag, fmt) con in the C and Python versions, the offscreen console handler or NULL for the root console x,y Coordinates of the rectangle's upper-left corner in the console. 0 <= x < console width 0 <= y < console height w,h size of the rectangle in the console. x <= x+w < console width y <= y+h < console height clear if true, all characters inside the rectangle are set to ASCII code 32 (space). If false, only the background color is modified flag this flag defines how the cell's background color is modified. See TCOD_bkgnd_flag_t fmt if NULL, the function only draws a rectangle. Else, printf-like format string, eventually followed by parameters. You can use control codes to change the colors inside the string.

int **getWidth**() **const**

console_read Reading the content of the console console_draw Get the console's width This function returns the width of a console (either the root console or an offscreen console) int *TCODConsole::getWidth() const* int TCOD_console_get_width(TCOD_console_t con) console_get_width(con) # int *TCODConsole::getWidth()* Console:*getWidth()* con in the C and Python versions, the offscreen console handler or NULL for the root console

int **getHeight**() **const**

console_read Get the console's height This function returns the height of a console (either the root console or an offscreen console) int *TCODConsole::getHeight() const* int TCOD_console_get_height(TCOD_console_t con) console_get_height(con) # int *TCODConsole::getHeight()* Console:*getHeight()* con in the C and Python versions, the offscreen console handler or NULL for the root console

TCODColor **getDefaultBackground**() **const**

console_read Reading the default background color This function returns the default background color of a console.

TCODColor TCODConsole::getDefaultBackground() const TCOD_color_t TCOD_console_get_default_background(TCOD_console_t con) console_get_default_background(con) # *TCODColor* TCODConsole::getBackgroundColor() Console:getBackgroundColor() con in the C and Python versions, the offscreen console handler or NULL for the root console

TCODColor **getDefaultForeground**() **const**

console_read Reading the default foreground color This function returns the default foreground color of a console.

TCODColor TCODConsole::getDefaultForeground() const TCOD_color_t TCOD_console_get_default_foreground(TCOD_console_t con) console_get_default_foreground(con) # *TCODColor* TCODConsole::getForegroundColor() Console:getForegroundColor() con in the C and Python versions, the offscreen console handler or NULL for the root console

TCODColor **getCharBackground**(int *x*, int *y*) **const**

console_read Reading the background color of a cell This function returns the background color of a cell.

TCODColor TCODConsole::getCharBackground(int x, int y) const TCOD_color_t TCOD_console_get_char_background(TCOD_console_t con,int x, int y) console_get_char_background(con,x,y) # *TCODColor* TCODConsole::getCharBackground(int x, int y) Console::getCharBackground(x, y) con in the C and Python versions, the offscreen console handler or

NULL for the root console x,y coordinates of the cell in the console. 0 <= x < console width 0 <= y < console height

TCODColor **getCharForeground** (int *x*, int *y*) **const**

console_read Reading the foreground color of a cell This function returns the foreground color of a cell.

TCODColor TCODConsole::getCharForeground(int x, int y) const TCOD_color_t TCOD_console_get_char_foreground(TCOD_console_t con,int x, int y) console_get_char_foreground(con,x,y) # *TCODColor* TCODConsole::getCharForeground(int x, int y) Console::getCharForeground(x, y) con in the C and Python versions, the offscreen console handler or NULL for the root console x,y coordinates of the cell in the console. 0 <= x < console width 0 <= y < console height

int **getChar** (int *x*, int *y*) **const**

console_read Reading the ASCII code of a cell This function returns the ASCII code of a cell.

int *TCODConsole::getChar(int x, int y) const* int TCOD_console_get_char(TCOD_console_t con,int x, int y) console_get_char(con,x,y) # int TCODConsole::getChar(int x, int y) Console::getChar(x, y) con in the C and Python versions, the offscreen console handler or NULL for the root console x,y coordinates of the cell in the console. 0 <= x < console width 0 <= y < console height

TCODConsole (int *w*, int *h*)

console_key_t Keyboard event structure console_input This structure contains information about a key pressed/released by the user.

typedef struct { TCOD_keycode_t vk; char c; char text[32]; bool pressed; bool lalt; bool lctrl; bool lmeta; bool ralt; bool rctrl; bool rmeta; bool shift; } *TCOD_key_t*; key.KeyCode key.Character key.Text key.Pressed key.LeftAlt key.LeftControl key.LeftMeta key.RightAlt key.RightControl key.RightMeta key.Shift vk An arbitrary value representing the physical key on the keyboard. Possible values are stored in the TCOD_keycode_t enum. If no key was pressed, the value is TCODK_NONE c If the key correspond to a printable character, the character is stored in this field. Else, this field contains 0. text If vk is TCODK_TEXT, this will contain the text entered by the user. pressed true if the event is a key pressed, or false for a key released. lalt This field represents the status of the left Alt key : true => pressed, false => released. lctrl This field represents the status of the left Control key : true => pressed, false => released. lmeta This field represents the status of the left Meta (Windows/Command/..) key : true => pressed, false => released. ralt This field represents the status of the right Alt key : true => pressed, false => released. rctrl This field represents the status of the right Control key : true => pressed, false => released. rmeta This field represents the status of the right Meta (Windows/Command/..) key : true => pressed, false => released. shift This field represents the status of the shift key : true => pressed, false => released. console_mouse_t Mouse event structure console_input This structure contains information about a mouse move/press/release event. typedef struct { int x,y; int dx,dy; int cx,cy; int dcx,dcy; bool lbutton; bool rbutton; bool mbutton; bool lbutton_pressed; bool rbutton_pressed; bool mbutton_pressed; bool wheel_up; bool wheel_down; } *TCOD_mouse_t*; x,y Absolute position of the mouse cursor in pixels relative to the window top-left corner. dx,dy Movement of the mouse cursor since the last call in pixels. cx,cy Coordinates of the console cell under the mouse cursor (pixel coordinates divided by the font size). dcx,dcy Movement of the mouse since the last call in console cells (pixel coordinates divided by the font size). lbutton true if the left button is pressed. rbutton true if the right button is pressed. mbutton true if the middle button (or the wheel) is pressed. lbutton_pressed true if the left button was pressed and released. rbutton_pressed true if the right button was pressed and released. mbutton_pressed true if the middle button was pressed and released. wheel_up true if the wheel was rolled up. wheel_down true if the wheel was rolled down. console_keycode_t Key codes console_input TCOD_keycode_t is a libtcod specific code representing a key on the keyboard. For Python, replace TCODK by KEY: libtcod.KEY_NONE. C# and Lua, the value is in parenthesis. Possible values are : When no key was pressed (see checkForEvent) : TCOD_NONE (NoKey) Special keys : TCODK_ESCAPE (Escape) TCODK_BACKSPACE (Backspace) TCODK_TAB (Tab) TCODK_ENTER (Enter) TCODK_SHIFT (Shift) TCODK_CONTROL (Control) TCODK_ALT (Alt) TCODK_PAUSE (Pause) TCODK_CAPSLOCK (CapsLock) TCODK_PAGEUP (PageUp) TCODK_PAGEDOWN

(PageDown) TCODK_END (End) TCODK_HOME (Home) TCODK_UP (Up) TCODK_LEFT (Left) TCODK_RIGHT (Right) TCODK_DOWN (Down) TCODK_PRINTSCREEN (Printscreen) TCODK_INSERT (Insert) TCODK_DELETE (Delete) TCODK_LWIN (Lwin) TCODK_RWIN (Rwin) TCODK_APPS (Apps) TCODK_KPADD (KeypadAdd) TCODK_KPSUB (KeypadSubtract) TCODK_KPDIV (KeypadDivide) TCODK_KPMUL (KeypadMultiply) TCODK_KPDEC (Keypad-Decimal) TCODK_KPENTER (KeypadEnter) TCODK_F1 (F1) TCODK_F2 (F2) TCODK_F3 (F3) TCODK_F4 (F4) TCODK_F5 (F5) TCODK_F6 (F6) TCODK_F7 (F7) TCODK_F8 (F8) TCODK_F9 (F9) TCODK_F10 (F10) TCODK_F11 (F11) TCODK_F12 (F12) TCODK_NUMLOCK (Numlock) TCODK_SCROLLLOCK (Scrolllock) TCODK_SPACE (Space)

numeric keys :

TCODK_0 (Zero) TCODK_1 (One) TCODK_2 (Two) TCODK_3 (Three) TCODK_4 (Four) TCODK_5 (Five) TCODK_6 (Six) TCODK_7 (Seven) TCODK_8 (Eight) TCODK_9 (Nine) TCODK_KP0 (KeypadZero) TCODK_KP1 (KeypadOne) TCODK_KP2 (KeypadTwo) TCODK_KP3 (KeypadThree) TCODK_KP4 (KeypadFour) TCODK_KP5 (KeypadFive) TCODK_KP6 (KeypadSix) TCODK_KP7 (KeypadSeven) TCODK_KP8 (KeypadEight) TCODK_KP9 (KeypadNine)

Any other (printable) key :

TCODK_CHAR (Char)

Codes starting with TCODK_KP represents keys on the numeric keypad (if available). console_offscreen console Using off-screen consoles The offscreen consoles allow you to draw on secondary consoles as you would do with the root console. You can then blit those secondary consoles on the root console. This allows you to use local coordinate space while rendering a portion of the final screen, and easily move components of the screen without modifying the rendering functions. Creating an offscreen console You can create as many off-screen consoles as you want by using this function. You can draw on them as you would do with the root console, but you cannot flush them to the screen. Else, you can blit them on other consoles, including the root console. See blit. The C version of this function returns a console handler that you can use in most console drawing functions. *TCODConsole::TCODConsole(int w, int h)* TCOD_console_t *TCOD_console_new(int w, int h)* console_new(w,h) # *TCODConsole::TCODConsole(int w, int h)* tcod.Console(w,h) w,h the console size. 0 < w 0 < h Creating a 40x20 offscreen console, filling it with red and blitting it on the root console at position 5,5 *TCODConsole* *offscreenConsole = new *TCODConsole(40,20)*; offscreenConsole->setDefaultBackground(TCODColor::red); offscreenConsole->*clear()*; TCODConsole::blit(offscreenConsole,0,0,40,20,TCODConsole::root,5,5,255); TCOD_console_t offscreen_console = TCOD_console_new(40,20); TCOD_console_set_default_background(offscreen_console,TCOD_red); TCOD_console_clear(offscreen_console); TCOD_console_blit(offscreen_console,0,0,40,20,NULL,5,5,255); offscreen_console = libtcod.console_new(40,20) libtcod.console_set_background_color(offscreen_console,libtcod.red) libtcod.console_clear(offscreen_console) libtcod.console_blit(offscreen_console,0,0,40,20,0,5,5,255) Creating a 40x20 offscreen console, filling it with red and blitting it on the root console at position 5,5 offscreenConsole = tcod.Console(40,20) offscreenConsole:setBackgroundColor(tcod.color.red) offscreenConsole:*clear()* tcod.console.blit(offscreenConsole,0,0,40,20,libtcod.TCODConsole_root,5,5,255)

TCODConsole(**const** char **filename*)

console_offscreen Creating an offscreen console from a .asc or .apf file You can create an offscreen console from a file created with Ascii Paint with this constructor *TCODConsole::TCODConsole(const char *filename)* TCOD_console_t TCOD_console_from_file(const char *filename) filename path to the .asc or .apf file created with Ascii Paint Creating an offscreen console, filling it with data from the .asc file *TCODConsole* *offscreenConsole = new *TCODConsole*(“myfile.asc”); TCOD_console_t offscreen_console = TCOD_console_from_file(“myfile.apf”);

bool **loadAsc**(**const** char **filename*)

console_offscreen Loading an offscreen console from a .asc file You can load data from a file created with Ascii Paint with this function.

When needed, the console will be resized to fit the file size. The function returns false if it couldn't read the file. bool *TCODConsole::loadAsc(const char *filename)* bool TCOD_console_load_asc(TCOD_console_t con, const char *filename) con in the C and Python versions, the offscreen console handler filename path to the .asc file created with Ascii Paint Creating a 40x20 offscreen console *TCODConsole* *offscreenConsole = new *TCODConsole(40,20)*; possibly resizing it and filling it with data from the .asc file offscreenConsole->loadAsc("myfile.asc"); TCOD_console_t offscreen_console = TCOD_console_new(40,20); TCOD_console_load_asc(offscreen_console,"myfile.asc");

bool **loadApf**(**const** char **filename*)

console_offscreen Loading an offscreen console from a .apf file You can load data from a file created with Ascii Paint with this function.

When needed, the console will be resized to fit the file size. The function returns false if it couldn't read the file. bool *TCODConsole::loadApf(const char *filename)* bool TCOD_console_load_apf(TCOD_console_t con, const char *filename) con in the C and Python versions, the offscreen console handler

filename path to the .apf file created with Ascii Paint

Creating a 40x20 offscreen console *TCODConsole* *offscreenConsole = new *TCODConsole(40,20)*; possibly resizing it and filling it with data from the .apf file offscreenConsole->loadApf("myfile.apf"); TCOD_console_t offscreen_console = TCOD_console_new(40,20); TCOD_console_load_apf(offscreen_console,"myfile.asc");

bool **saveAsc**(**const** char **filename*) **const**

console_offscreen Saving a console to a .asc file You can save data from a console to Ascii Paint format with this function.

The function returns false if it couldn't write the file. This is the only ASC function that works also with the root console ! bool *TCODConsole::saveAsc(const char *filename) const* bool TCOD_console_save_asc(TCOD_console_t con, const char *filename) con in the C and Python versions, the offscreen console handler or NULL for the root console filename path to the .asc file to be created console->saveAsc("myfile.asc"); TCOD_console_save_asc(console,"myfile.asc");

bool **saveApf**(**const** char **filename*) **const**

console_offscreen Saving a console to a .apf file You can save data from a console to Ascii Paint format with this function.

The function returns false if it couldn't write the file. This is the only ASC function that works also with the root console ! bool *TCODConsole::saveApf(const char *filename) const* bool TCOD_console_save_apf(TCOD_console_t con, const char *filename) con in the C and Python versions, the offscreen console handler or NULL for the root console filename path to the .apf file to be created console->saveApf("myfile.apf"); TCOD_console_save_apf(console,"myfile.apf");

void **setKeyColor**(**const** *TCODColor* &*col*)

console_offscreen Define a blit-transparent color This function defines a transparent background color for an offscreen console.

All cells with this background color are ignored by the blit operation. You can use it to blit only some parts of the console. void *TCODConsole::setKeyColor(const TCODColor &col)* void TCOD_console_set_key_color(TCOD_console_t con,TCOD_color_t col) console_set_key_color(con,col) # void TCODConsole::setKeyColor(TCODColor col) Console:setKeyColor(col) con in the C and Python versions, the offscreen console handler or NULL for the root console col the transparent background color

Public Static Functions

void **initRoot**(int *w*, int *h*, **const** char **title*, bool *fullscreen* = false, *TCOD_renderer_t renderer* = *TCOD_RENDERER_SDL*)

console_init_root Creating the game window console_init static void *TCODConsole::initRoot* (int w, int h, const char * title, bool fullscreen = false, TCOD_renderer_t renderer = TCOD_RENDERER_SDL) void TCOD_console_init_root (int w, int h, const char * title, bool fullscreen, TCOD_renderer_t renderer) console_init_root (w, h, title, fullscreen = False, renderer = RENDERER_SDL) # static void TCODConsole::initRoot(int w, int h, string title) static void TCODConsole::initRoot(int w, int h, string title, bool fullscreen) static void TCODConsole::initRoot(int w, int h, string title, bool fullscreen, TCODRendererType renderer) tcod.console.initRoot(w,h,title) fullscreen = false, renderer = SDL tcod.console.initRoot(w,h,title,fullscreen) renderer = SDL tcod.console.initRoot(w,h,title,fullscreen,renderer) renderers : tcod.GLSL, tcod.OpenGL, tcod.SDL w,h size of the console(in characters).

The default font in libtcod (./terminal.png) uses 8x8 pixels characters. You can change the font by calling *TCODConsole::setCustomFont* before calling initRoot. title title of the window. It's not visible when you are in fullscreen. Note 1 : you can dynamically change the window title with *TCODConsole::setWindowTitle* fullscreen whether you start in windowed or fullscreen mode. Note 1 : you can dynamically change this mode with *TCODConsole::setFullscreen* Note 2 : you can get current mode with *TCODConsole::isFullscreen* renderer which renderer to use. Possible values are : TCOD_RENDERER_GLSL : works only on video cards with pixel shaders TCOD_RENDERER_OPENGL : works on all video cards supporting OpenGL 1.4 TCOD_RENDERER_SDL : should work everywhere! Note 1: if you select a renderer that is not supported by the player's machine, libtcod scan the lower renderers until it finds a working one. Note 2: on recent video cards, GLSL results in up to 900% increase of framerates in the true color sample compared to SDL renderer. Note 3: whatever renderer you use, it can always be overridden by the player through the libtcod.cfg file. Note 4: you can dynamically change the renderer after calling initRoot with *TCODSystem::setRenderer*. Note 5: you can get current renderer with *TCODSystem::getRenderer*. It might be different from the one you set in initRoot in case it's not supported on the player's computer. *TCODConsole::initRoot*(80, 50, "The Chronicles Of Doryen v0.1"); TCOD_console_init_root(80, 50, "The Chronicles Of Doryen v0.1", false, TCOD_RENDERER_OPENGL); libtcod.console_init_root(80, 50, 'The Chronicles Of Doryen v0.1') tcod.console.initRoot(80,50,"The Chronicles Of Doryen v0.1")

void **setCustomFont**(**const** char **fontFile*, int *flags* = *TCOD_FONT_LAYOUT_ASCII_INCOL*, int *nbCharHoriz* = 0, int *nbCharVertic* = 0)

console_set_custom_font Using a custom bitmap font console_init setCustomFont This function allows you to use a bitmap font (png or bmp) with custom character size or layout.

It should be called before initializing the root console with initRoot. Once this function is called, you can define your own custom mappings using mapping functions *Different font layouts*

- ascii, in columns : characters 0 to 15 are in the first column. The space character is at coordinates 2,0.
- ascii, in rows : characters 0 to 15 are in the first row. The space character is at coordinates 0,2.
- tcod : special mapping. Not all ascii values are mapped. The space character is at coordinates 0,0.

Different font types

- standard : transparency is given by a key color automatically detected by looking at the color of the space character
- 32 bits : transparency is given by the png alpha layer. The font color does not matter but it must be desaturated

- greyscale : transparency is given by the pixel value. You can use white characters on black background or black characters on white background. The background color is automatically detected by looking at the color of the space character

Examples of fonts can be found in libtcod's fonts directory. Check the Readme file there. static void *TCODConsole::setCustomFont*(const char *fontFile, int flags=TCOD_FONT_LAYOUT_ASCII_INCOL,int nbCharHoriz=0, int nbCharVertic=0) void TCOD_console_set_custom_font(const char *fontFile, int flags,int nb_char_horiz, int nb_char_vertic) console_set_custom_font(fontFile, flags=FONT_LAYOUT_ASCII_INCOL,nb_char_horiz=0, nb_char_vertic=0) # static void TCODConsole::setCustomFont(string fontFile) static void TCODConsole::setCustomFont(string fontFile, int flags) static void TCODConsole::setCustomFont(string fontFile, int flags, int nbCharHoriz) static void TCODConsole::setCustomFont(string fontFile, int flags, int nbCharHoriz, int nbCharVertic) tcod.console.setCustomFont(fontFile) tcod.console.setCustomFont(fontFile, flags) tcod.console.setCustomFont(fontFile, nbCharHoriz) tcod.console.setCustomFont(fontFile, flags, nbCharHoriz, nbCharVertic) flags : tcod.LayoutAsciiInColumn, tcod.LayoutAsciiInRow, tcod.LayoutTCOD, tcod.Greyscale fontFile Name of a .bmp or .png file containing the font. flags Used to define the characters layout in the bitmap and the font type : TCOD_FONT_LAYOUT_ASCII_INCOL : characters in ASCII order, code 0-15 in the first column TCOD_FONT_LAYOUT_ASCII_INROW : characters in ASCII order, code 0-15 in the first row TCOD_FONT_LAYOUT_TCOD : simplified layout. See examples below. TCOD_FONT_TYPE_GREYSCALE : create an anti-aliased font from a greyscale bitmap For Python, remove TCOD _ : libtcod.FONT_LAYOUT_ASCII_INCOL nbCharHoriz,nbCharVertic Number of characters in the font. Should be 16x16 for ASCII layouts, 32x8 for TCOD layout. But you can use any other layout. If set to 0, there are deduced from the font layout flag. *TCODConsole::setCustomFont*("standard_8x8_ascii_in_col_font.bmp",TCOD_FONT_LAYOUT_ASCII_INCOL); *TCODConsole::setCustomFont*("32bits_8x8_ascii_in_row_font.png",TCOD_FONT_LAYOUT_ASCII_INROW); *TCODConsole::setCustomFont*("greyscale_8x8_tcod_font.png",TCOD_FONT_LAYOUT_TCOD | TCOD_FONT_TYPE_GREYSCALE); TCOD_console_set_custom_font("standard_8x8_ascii_in_col_font.bmp",TCOD_F
TCOD_console_set_custom_font("32bits_8x8_ascii_in_row_font.png",TCOD_FONT_LAYOUT_ASCII_INROW,32,8); TCOD_console_set_custom_font("greyscale_8x8_tcod_font.png",TCOD_FONT_LAYOUT_TCOD | TCOD_FONT_TYPE_GREYSCALE,32,8); libtcod.console_set_custom_font("standard_8x8_ascii_in_col_font.bmp",libtco
libtcod.console_set_custom_font("32bits_8x8_ascii_in_row_font.png",libtcod.FONT_LAYOUT_ASCII_INROW) libtcod.console_set_custom_font("greyscale_8x8_tcod_font.png",libtcod.FONT_LAYOUT_TCOD | libtcod.FONT_TYPE_GREYSCALE) tcod.console.setCustomFont("standard_8x8_ascii_in_col_font.bmp",tcod.LayoutAsciiIn
tcod.console.setCustomFont("32bits_8x8_ascii_in_row_font.png",tcod.LayoutAsciiInRow); tcod.console.setCustomFont("greyscale_8x8_tcod_font.png",tcod.LayoutTCOD + tcod.Greyscale);

void **mapAsciiCodeToFont** (int *asciiCode*, int *fontCharX*, int *fontCharY*)

console_map Using custom characters mapping console_init Mapping a single ASCII code to a character These functions allow you to map characters in the bitmap font to ASCII codes.

They should be called after initializing the root console with initRoot. You can dynamically change the characters mapping at any time, allowing to use several fonts in the same screen. static void *TCODConsole::mapAsciiCodeToFont(int asciiCode, int fontCharX, int fontCharY)* void TCOD_console_map_ascii_code_to_font(int asciiCode, int fontCharX, int fontCharY) console_map_ascii_code_to_font(asciiCode, fontCharX, fontCharY) # static void *TCODConsole::mapAsciiCodeToFont(int asciiCode, int fontCharX, int fontCharY)* tcod.console.mapAsciiCodeToFont(asciiCode, fontCharX, fontCharY) asciiCode ASCII code to map. fontCharX,fontCharY Coordinate of the character in the bitmap font (in characters, not pixels).

void **mapAsciiCodesToFont** (int *firstAsciiCode*, int *nbCodes*, int *fontCharX*, int *fontCharY*)

console_map Mapping consecutive ASCII codes to consecutive characters static void *TCODConsole::mapAsciiCodesToFont(int firstAsciiCode, int nbCodes, int fontCharX, int fontCharY)* void TCOD_console_map_ascii_codes_to_font(int firstAsciiCode, int nbCodes, int fontCharX, int fontCharY) console_map_ascii_codes_to_font(firstAsciiCode, nbCodes, fontCharX, fontCharY) # static void

TCODConsole::mapAsciiCodesToFont(int firstAsciiCode, int nbCodes, int fontCharX, int fontCharY) tcod.console.mapAsciiCodesToFont(firstAsciiCode, nbCodes, fontCharX, fontCharY) firstAsciiCode first ASCII code to map nbCodes number of consecutive ASCII codes to map fontCharX,fontCharY coordinate of the character in the bitmap font (in characters, not pixels) corresponding to the first ASCII code

void **mapStringToFont**(**const** char *s, int *fontCharX*, int *fontCharY*)

console_map Mapping ASCII code from a string to consecutive characters static void *TCODConsole::mapStringToFont(const char *s, int fontCharX, int fontCharY)* void TCOD_console_map_string_to_font(const char *s, int fontCharX, int fontCharY) console_map_string_to_font(s, fontCharX, fontCharY) # static void TCODConsole::mapStringToFont(string s, int fontCharX, int fontCharY) tcod.console.mapStringToFont(s, fontCharX, fontCharY) s string containing the ASCII codes to map fontCharX,fontCharY coordinate of the character in the bitmap font (in characters, not pixels) corresponding to the first ASCII code in the string

bool **isFullscreen**()

console_fullscreen Fullscreen mode console_init Getting the current mode This function returns true if the current mode is fullscreen.

static bool *TCODConsole::isFullscreen()* bool TCOD_console_is_fullscreen() console_is_fullscreen() # static bool *TCODConsole::isFullscreen()* tcod.console.isFullscreen()

void **setFullscreen**(bool *fullscreen*)

console_fullscreen Switching between windowed and fullscreen modes This function switches the root console to fullscreen or windowed mode.

Note that there is no predefined key combination to switch to/from fullscreen. You have to do this in your own code. static void *TCODConsole::setFullscreen(bool fullscreen)* void *TCOD_console_set_fullscreen(bool fullscreen)* console_set_fullscreen(fullscreen) # static void *TCODConsole::setFullscreen(bool fullscreen)* tcod.console.setFullscreen(fullscreen) fullscreen true to switch to fullscreen mode. false to switch to windowed mode. *TCOD_key_t* key; TCODConsole::checkForEvent(TCOD_EVENT_KEY_PRESS,&key,NULL); if (key.vk == TCODK_ENTER && key.lalt) *TCODConsole::setFullscreen*(!TCODConsole::isFullscreen()); *TCOD_key_t* key; TCOD_console_check_for_event(TCOD_EVENT_KEY_PRESS,&key,NULL); if (key.vk == TCODK_ENTER && key.lalt) TCOD_console_set_fullscreen(!TCOD_console_is_fullscreen()); key=Key() libtcod.console_check_for_event(libtcod.EVENT_KEY_PRESS,key,0) if key.vk == libtcod.KEY_ENTER and key.lalt : libtcod.console_set_fullscreen(not libtcod.console_is_fullscreen()) key=tcod.console.checkForKeypress() if key.KeyCode == tcod.Enter and key.LeftAlt then tcod.console.setFullscreen(not tcod.console.isFullscreen()) end

void **setWindowTitle**(**const** char **title*)

console_window console_init Communicate with the window manager Changing the window title This function dynamically changes the title of the game window.

Note that the window title is not visible while in fullscreen. static void *TCODConsole::setWindowTitle(const char *title)* void *TCOD_console_set_window_title(const char *title)* console_set_window_title(title) # static void TCODConsole::setWindowTitle(string title) tcod.console.setWindowTitle(title) title New title of the game window

bool **isWindowClosed**()

console_window Handling “close window” events When you start the program, this returns false.

Once a “close window” event has been sent by the window manager, it will always return true. You’re supposed to exit cleanly the game. static bool *TCODConsole::isWindowClosed()* bool TCOD_console_is_window_closed() console_is_window_closed() # static bool *TCODConsole::isWindowClosed()* tcod.console.isWindowClosed()

bool **hasMouseFocus**()

console_window Check if the mouse cursor is inside the game window Returns true if the mouse cursor is inside the game window area and the game window is the active application.

static bool *TCODConsole::hasMouseFocus()* bool TCOD_console_has_mouse_focus() console_has_mouse_focus()

bool **isActive**()

console_window Check if the game application is active Returns false if the game window is not the active window or is iconified.

static bool *TCODConsole::isActive()* bool TCOD_console_is_active() console_is_active()

void **credits**()

console_credits libtcod's credits console_init Use these functions to display credits, as seen in the samples.

Using a separate credit page You can print a "Powered by libtcod x.y.z" screen during your game startup simply by calling this function after initRoot. The credits screen can be skipped by pressing any key. static void *TCODConsole::credits()* void *TCOD_console_credits()* console_credits() # static void *TCODConsole::credits()* tcod.console.credits()

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

console_credits Embedding credits in an existing page You can also print the credits on one of your game screens (your main menu for example) by calling this function in your main loop.

This function returns true when the credits screen is finished, indicating that you no longer need to call it. static bool *TCODConsole::renderCredits(int x, int y, bool alpha)* bool TCOD_console_credits_render(int x, int y, bool alpha) bool TCOD_console_credits_render(int x, int y, bool alpha) # static bool *TCODConsole::renderCredits(int x, int y, bool alpha)* tcod.console.renderCredits(x, y, alpha) x,y Position of the credits text in your root console alpha If true, credits are transparently added on top of the existing screen. For this to work, this function must be placed between your screen rendering code and the console flush. *TCODConsole::initRoot*(80,50,"The Chronicles Of Doryen v0.1",false); // initialize the root console bool endCredits=false; while (! *TCODConsole::isWindowClosed()*) { // your game loop your game rendering here… render transparent credits near the center of the screen if (! endCredits) endCredits=TCODConsole::renderCredits(35,25,true); *TCODConsole::flush()*; } TCOD_console_init_root(80,50,"The Chronicles Of Doryen v0.1",false); bool end_credits=false; while (! TCOD_console_is_window_closed()) { your game rendering here… render transparent credits near the center of the screen if (! end_credits) end_credits=TCOD_console_credits_render(35,25,true); *TCOD_console_flush()*; } libtcod.console_init_root(80,50,"The Chronicles Of Doryen v0.1",False) end_credits=False while not libtcod.console_is_window_closed() : your game rendering here… render transparent credits near the center of the screen if (not end_credits) : end_credits=libtcod.console_credits_render(35,25,True) libtcod.console_flush() tcod.console.initRoot(80,50,"The Chronicles Of Doryen v0.1") initialize the root console endCredits=false while not tcod.console.isWindowClosed() do your game loop your game rendering here… render transparent credits near the center of the screen if not endCredits then endCredits=tcod.console.renderCredits(35,25,true) end tcod.console.flush() end

void **resetCredits**()

console_credits Restart the credits animation When using rederCredits, you can restart the credits animation from the beginning before it's finished by calling this function.

static void *TCODConsole::resetCredits()* void *TCOD_console_credits_reset()* console_credits_reset() # static void *TCODConsole::resetCredits()* tcod.console.resetCredits()

void **setColorControl**(TCOD_colctrl_t *con*, **const** *TCODColor* *&fore*, **const** *TCODColor* *&back*)

console_print Changing the colors while printing a string If you want to draw a string using different colors for each word, the basic solution is to call a string printing function several times, changing the default colors between each call.

The TCOD library offers a simpler way to do this, allowing you to draw a string using different colors in a single call. For this, you have to insert color control codes in your string. A color control code is associated with a color set (a foreground color and a background color). If you insert this code in your string, the next characters will use the colors associated with the color control code. There are 5 predefined color control codes : For Python, remove TCOD_ : libtcod.COLCTRL_1 TCOD_COLCTRL_1 TCOD_COLCTRL_2 TCOD_COLCTRL_3 TCOD_COLCTRL_4 TCOD_COLCTRL_5 To associate a color with a code, use setColorControl. To go back to the console's default colors, insert in your string the color stop control code : TCOD_COLCTRL_STOP

You can also use any color without assigning it to a control code, using the generic control codes : TCOD_COLCTRL_FORE_RGB TCOD_COLCTRL_BACK_RGB

Those controls respectively change the foreground and background color used to print the string characters. In the string, you must insert the r,g,b components of the color (between 1 and 255. The value 0 is forbidden because it represents the end of the string in C/C++) immediately after this code. static void *TCODConsole::setColorControl(TCOD_colctrl_t con, const TCODColor &fore, const TCODColor &back)* void TCOD_console_set_color_control(TCOD_colctrl_t con, TCOD_color_t fore, TCOD_color_t back) console_set_color_control(con,fore,back) # Not supported directly, use getRGBColorControlString and getColorControlString. Not supported con the color control TCOD_COLCTRL_x, 1<=x<=5 fore foreground color when this control is activated back background color when this control is activated A string with a red over black word, using predefined color control codes TCODConsole::setColorControl(TCOD_COLCTRL_1,TCODColor::red,TCODColor::black); *TCODConsole::root*->print(1,1,"String with a %cred%c word.",TCOD_COLCTRL_1,TCOD_COLCTRL_STOP); A string with a red over black word, using generic color control codes *TCODConsole::root*->print(1,1,"String with a %c%c%c%c%c%c%c%cred%c word.", TCOD_COLCTRL_FORE_RGB,255,1,1,TCOD_COLCTRL_BACK_RGB,1,1,1,TCOD_COLCTRL_STOP); A string with a red over black word, using generic color control codes *TCODConsole::root*->print(1,1,"String with a %c%c%c%c%c%c%c%cred%c word.", TCOD_COLCTRL_FORE_RGB,255,1,1,TCOD_COLCTRL_BACK_RGB,1,1,1,TCOD_COLCTRL_STOP); A string with a red over black word, using predefined color control codes TCOD_console_set_color_control(TCOD_COLCTRL_1,red,black); TCOD_console_print(NULL,1,1,"String with a %cred%c word.",TCOD_COLCTRL_1,TCOD_COLCTRL_STOP); A string with a red word (over default background color), using generic color control codes TCOD_console_print(NULL,1,1,"String with a %c%c%c%cred%c word.", TCOD_COLCTRL_FORE_RGB,255,1,1,TCOD_COLCTRL_STOP); A string with a red over black word, using generic color control codes TCOD_console_print(NULL,1,1,"String with a %c%c%c%c%c%c%c%cred%c word.", TCOD_COLCTRL_FORE_RGB,255,1,1,TCOD_COLCTRL_BACK_RGB,1,1,1,TCOD_COLCTRL_STOP);
A string with a red over black word, using predefined color control codes

libtcod.console_set_color_control(libtcod.COLCTRL_1,litbcod.red,litbcod.black) libtcod.console_print(0,1,1,"String with a %cred%c word."%(libtcod.COLCTRL_1,libtcod.COLCTRL_STOP))
A string with a red word (over default background color), using generic color control codes

litbcod.console_print(0,1,1,"String with a %c%c%c%cred%c word."%(libtcod.COLCTRL_FORE_RGB,255,1,1,libtcod.CO
A string with a red over black word, using generic color control codes

libtcod.console_print(0,1,1,"String with a %c%c%c%c%c%c%c%cred%c word."% (libtcod.COLCTRL_FORE_RGB,255,1,1,libtcod.COLCTRL_BACK_RGB,1,1,1,libtcod.COLCTRL_STOP))

#Ex TCODConsole.root.print(1,1,String.Format("String with a {0}red{1} word.", TCODConsole.getRGBColorControlString(ColorControlForeground,TCODColor.red), TCODConsole.getColorControlString(ColorControlStop));

void **mapStringToFont** (**const** wchar_t **s*, int *fontCharX*, int *fontCharY*)

console_print Unicode functions those functions are similar to their ASCII equivalent, but work with unicode strings (wchar_t in C/C++).

Note that unicode is not supported in the Python wrapper. static void *TCODConsole::mapStringToFont(const wchar_t *s, int fontCharX, int fontCharY)* void TCOD_console_map_string_to_font_utf(const wchar_t *s, int fontCharX, int fontCharY)

void **setFade** (uint8_t *fade*, **const** *TCODColor* *&fadingColor*)

console_fading Screen fading functions console_draw Use these functions to easily fade to/from a color Changing the fading parameters This function defines the fading parameters, allowing to easily fade the game screen to/from a color.

Once they are defined, the fading parameters are valid for ever. You don't have to call setFade for each rendered frame (unless you change the fading parameters). static void *TCODConsole::setFade(uint8_t fade, const TCODColor &fadingColor)* void *TCOD_console_set_fade(uint8_t fade, TCOD_color_t fadingColor)* console_set_fade(fade, fadingColor) # static void TCODConsole::setFade(byte fade, TCODColor fadingColor) tcod.console.setFade(fade, fadingColor) fade the fading amount. 0 => the screen is filled with the fading color. 255 => no fading effect fadingColor the color to use during the console flushing operation for (int fade=255; fade >= 0; fade) { TCODConsole::setFade(fade,TCODColor::black); *TCODConsole::flush()*; } int fade; for (fade=255; fade >= 0; fade) { TCOD_console_setFade(fade,TCOD_black); *TCOD_console_flush()*; } for fade in range(255,0) : libtcod.console_setFade(fade,libtcod.black) libtcod.console_flush() for fade=255,0,-1 do tcod.console.setFade(fade,tcod.color.black) tcod.console.flush() end

uint8_t **getFade** ()

console_fading Reading the fade amount This function returns the current fade amount, previously defined by setFade.

static uint8_t *TCODConsole::getFade()* uint8_t TCOD_console_get_fade() console_get_fade() # static byte *TCODConsole::getFade()* tcod.console.getFade()

TCODColor **getFadingColor** ()

console_fading Reading the fading color This function returns the current fading color, previously defined by setFade.

static *TCODColor TCODConsole::getFadingColor()* TCOD_color_t TCOD_console_get_fading_color() console_get_fading_color() # static *TCODColor* *TCODConsole::getFadingColor()* tcod.console.getFadingColor()

void **flush** ()

console_flush console Flushing the root console Once the root console is initialized, you can use one of the printing functions to change the background colors, the foreground colors or the ASCII characters on the console.

Once you've finished rendering the root console, you have to actually apply the updates to the screen with this function. static void *TCODConsole::flush()* void *TCOD_console_flush()* console_flush() # static void *TCODConsole::flush()* tcod.console.flush()

TCOD_key_t **waitForKeypress** (bool *flush*)

console_ascii ASCII constants console_draw Some useful graphic characters in the terminal.bmp font.

For the Python version, remove TCOD_ from the constants. C# and Lua is in parenthesis : Single line walls: TCOD_CHAR_HLINE=196 (HorzLine) TCOD_CHAR_VLINE=179 (VertLine) TCOD_CHAR_NE=191 (NE) TCOD_CHAR_NW=218 (NW) TCOD_CHAR_SE=217 (SE) TCOD_CHAR_SW=192 (SW)

Double lines walls: TCOD_CHAR_DHLINE=205 (DoubleHorzLine) TCOD_CHAR_DVLINE=186 (DoubleVertLine) TCOD_CHAR_DNE=187 (DoubleNE) TCOD_CHAR_DNW=201 (DoubleNW)

TCOD_CHAR_DSE=188 (DoubleSE) TCOD_CHAR_DSW=200 (DoubleSW)

Single line vertical/horizontal junctions (T junctions): TCOD_CHAR_TEEW=180 (TeeWest) TCOD_CHAR_TEEE=195 (TeeEast) TCOD_CHAR_TEEN=193 (TeeNorth) TCOD_CHAR_TEES=194 (TeeSouth)

Double line vertical/horizontal junctions (T junctions): TCOD_CHAR_DTEEW=185 (DoubleTeeWest) TCOD_CHAR_DTEEE=204 (DoubleTeeEast) TCOD_CHAR_DTEEN=202 (DoubleTeeNorth) TCOD_CHAR_DTEES=203 (DoubleTeeSouth)

Block characters: TCOD_CHAR_BLOCK1=176 (Block1) TCOD_CHAR_BLOCK2=177 (Block2) TCOD_CHAR_BLOCK3=178 (Block3)

Cross-junction between two single line walls: TCOD_CHAR_CROSS=197 (Cross)

Arrows: TCOD_CHAR_ARROW_N=24 (ArrowNorth) TCOD_CHAR_ARROW_S=25 (ArrowSouth) TCOD_CHAR_ARROW_E=26 (ArrowEast) TCOD_CHAR_ARROW_W=27 (ArrowWest)

Arrows without tail: TCOD_CHAR_ARROW2_N=30 (ArrowNorthNoTail) TCOD_CHAR_ARROW2_S=31 (ArrowSouthNoTail) TCOD_CHAR_ARROW2_E=16 (ArrowEastNoTail) TCOD_CHAR_ARROW2_W=17 (ArrowWestNoTail)

Double arrows: TCOD_CHAR_DARROW_H=29 (DoubleArrowHorz) TCOD_CHAR_ARROW_V=18 (DoubleArrowVert)

GUI stuff: TCOD_CHAR_CHECKBOX_UNSET=224 TCOD_CHAR_CHECKBOX_SET=225 TCOD_CHAR_RADIO_UNSET=9 TCOD_CHAR_RADIO_SET=10

Sub-pixel resolution kit: TCOD_CHAR_SUBP_NW=226 (SubpixelNorthWest) TCOD_CHAR_SUBP_NE=227 (SubpixelNorthEast) TCOD_CHAR_SUBP_N=228 (SubpixelNorth) TCOD_CHAR_SUBP_SE=229 (SubpixelSouthEast) TCOD_CHAR_SUBP_DIAG=230 (SubpixelDiagonal) TCOD_CHAR_SUBP_E=231 (SubpixelEast) TCOD_CHAR_SUBP_SW=232 (SubpixelSouthWest)

Miscellaneous characters: TCOD_CHAR_SMILY = 1 (Smilie) TCOD_CHAR_SMILY_INV = 2 (SmilieInv) TCOD_CHAR_HEART = 3 (Heart) TCOD_CHAR_DIAMOND = 4 (Diamond) TCOD_CHAR_CLUB = 5 (Club) TCOD_CHAR_SPADE = 6 (Spade) TCOD_CHAR_BULLET = 7 (Bullet) TCOD_CHAR_BULLET_INV = 8 (BulletInv) TCOD_CHAR_MALE = 11 (Male) TCOD_CHAR_FEMALE = 12 (Female) TCOD_CHAR_NOTE = 13 (Note) TCOD_CHAR_NOTE_DOUBLE = 14 (NoteDouble) TCOD_CHAR_LIGHT = 15 (Light) TCOD_CHAR_EXCLAM_DOUBLE = 19 (ExclamationDouble) TCOD_CHAR_PILCROW = 20 (Pilcrow) TCOD_CHAR_SECTION = 21 (Section) TCOD_CHAR_POUND = 156 (Pound) TCOD_CHAR_MULTIPLICATION = 158 (Multiplication) TCOD_CHAR_FUNCTION = 159 (Function) TCOD_CHAR_RESERVED = 169 (Reserved) TCOD_CHAR_HALF = 171 (Half) TCOD_CHAR_ONE_QUARTER = 172 (OneQuarter) TCOD_CHAR_COPYRIGHT = 184 (Copyright) TCOD_CHAR_CENT = 189 (Cent) TCOD_CHAR_YEN = 190 (Yen) TCOD_CHAR_CURRENCY = 207 (Currency) TCOD_CHAR_THREE_QUARTERS = 243 (ThreeQuarters) TCOD_CHAR_DIVISION = 246 (Division) TCOD_CHAR_GRADE = 248 (Grade) TCOD_CHAR_UMLAUT = 249 (Umlaut) TCOD_CHAR_POW1 = 251 (Pow1) TCOD_CHAR_POW3 = 252 (Pow2) TCOD_CHAR_POW2 = 253 (Pow3) TCOD_CHAR_BULLET_SQUARE = 254 (BulletSquare) console_input Handling user input The user handling functions allow you to get keyboard and mouse input from the user, either for turn by turn games (the function wait until the user press a key or a mouse button), or real time games (non blocking function). **WARNING : those functions also handle screen redraw event, so *TCODConsole::flush* function won't redraw screen if no user input function is called !** console

bool **isKeyPressed**(*TCOD_keycode_t key*)

console_blocking_input Core console_input Blocking user input This function stops the application until an event occurs.

console_non_blocking_input Non blocking user input console_input The preferred way to check for user input is to use checkForEvent below, but you can also get the status of any special key at any time with : static bool *TCODConsole::isKeyPressed(TCOD_keycode_t key)* bool TCOD_console_is_key_pressed(TCOD_keycode_t key) console_is_key_pressed(key) # static bool TCODConsole::isKeyPressed(TCODKeyCode key) tcod.console.isKeyPressed(key) key Any key code defined in keycode_t except TCODK_CHAR (Char) and TCODK_NONE (NoKey)

void **blit** (**const** *TCODConsole* **src*, int *xSrc*, int *ySrc*, int *wSrc*, int *hSrc*, *TCODConsole* **dst*, int *xDst*, int *yDst*, float *foreground_alpha* = 1.0f, float *background_alpha* = 1.0f)

console_offscreen Blitting a console on another one This function allows you to blit a rectangular area of the source console at a specific position on a destination console.

It can also simulate alpha transparency with the fade parameter. static void blit(const *TCODConsole* *src,int xSrc, int ySrc, int wSrc, int hSrc, *TCODConsole* *dst, int xDst, int yDst, float foregroundAlpha=1.0f, float backgroundAlpha=1.0f) 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) console_blit(src,xSrc,ySrc,xSrc,hSrc,dst,xDst,yDst,foregroundAlpha=1.0,backgroundAlpha=1.0) # static void TCODConsole::blit(TCODConsole src, int xSrc, int ySrc, int wSrc, int hSrc, TCODConsole dst, int xDst, int yDst) static void TCODConsole::blit(TCODConsole src, int xSrc, int ySrc, int wSrc, int hSrc, TCODConsole dst, int xDst, int yDst, float foreground_alpha) static void TCODConsole::blit(TCODConsole src, int xSrc, int ySrc, int wSrc, int hSrc, TCODConsole dst, int xDst, int yDst, float foreground_alpha, float background_alpha) tcod.console.blit(src, xSrc, ySrc, wSrc, hSrc, dst, xDst, yDst) tcod.console.blit(src, xSrc, ySrc, wSrc, hSrc, dst, xDst, yDst, foreground_alpha) tcod.console.blit(src, xSrc, ySrc, wSrc, hSrc, dst, xDst, yDst, foreground_alpha, background_alpha) src The source console that must be blitted on another one. xSrc,ySrc,wSrc,hSrc The rectangular area of the source console that will be blitted. If wSrc and/or hSrc == 0, the source console width/height are used dst The destination console. xDst,yDst Where to blit the upper-left corner of the source area in the destination console. foregroundAlpha,backgroundAlpha Alpha transparency of the blitted console. 0.0 => The source console is completely transparent. This function does nothing. 1.0 => The source console is opaque. Its cells replace the destination cells. 0 < fade < 1.0 => The source console is partially blitted, simulating real transparency. Cross-fading between two offscreen consoles. We use two offscreen consoles with the same size as the root console. We render a different screen on each offscreen console. When the user hits a key, we do a cross-fading from the first screen to the second screen. *TCODConsole* *off1 = new *TCODConsole(80,50)*; *TCODConsole* *off2 = new *TCODConsole(80,50)*; . . . print screen1 on off1 . . . print screen2 of off2 render screen1 in the game window TCODConsole::blit(off1,0,0,80,50,TCODConsole::root,0,0); *TCODConsole::flush()*; wait or a keypress TCODConsole::waitForKeypress(true); do a cross-fading from off1 to off2 for (int i=1; i <= 255; i++) { TCODConsole::blit(off1,0,0,80,50,TCODConsole::root,0,0); // renders the first screen (opaque) *TCODConsole::blit*(off2,0,0,80,50,*TCODConsole::root*,0,0,i/255.0,i/255.0); // renders the second screen (transparent) *TCODConsole::flush()*; } TCOD_console_t off1 = TCOD_console_new(80,50); TCOD_console_t off2 = TCOD_console_new(80,50); int i; . . . print screen1 on off1 . . . print screen2 of off2 render screen1 in the game window TCOD_console_blit(off1,0,0,80,50,NULL,0,0,1.0,1.0); *TCOD_console_flush()*; wait or a keypress TCOD_console_wait_for_keypress(true); do a cross-fading from off1 to off2 for (i=1; i <= 255; i++) { TCOD_console_blit(off1,0,0,80,50,NULL,0,0,1.0,1.0); // renders the first screen (opaque) TCOD_console_blit(off2,0,0,80,50,NULL,0,0,i/255.0,i/255.0); // renders the second screen (transparent) *TCOD_console_flush()*; } off1 = libtcod.console_new(80,50) off2 = libtcod.console_new(80,50) . . . print screen1 on off1 . . . print screen2 of off2 *render screen1 in the game window*

libtcod.console_blit(off1,0,0,80,50,0,0,0) libtcod.console_flush() *wait or a keypress*

libtcod.console_wait_for_keypress(True) *do a cross-fading from off1 to off2*

for i in range(1,256) : litbcod.console_blit(off1,0,0,80,50,0,0,0) # renders the first screen (opaque) litbcod.console_blit(off2,0,0,80,50,0,0,0,i/255.0,i/255.0) # renders the second screen (transparent) litbcod.console_flush() Cross-fading between two offscreen consoles. We use two offscreen consoles with the same size as the root console. We render a different screen on each offscreen console.

When the user hits a key, we do a cross-fading from the first screen to the second screen. off1 = tcod.Console(80,50) off2 = tcod.Console(80,50) … print screen1 on off1 … print screen2 of off2 render screen1 in the game window root=libtcod.TCODConsole_root tcod.console.blit(off1,0,0,80,50,root,0,0) tcod.console.flush() wait or a keypress tcod.console.waitForKeypress(true) do a cross-fading from off1 to off2 for i=1,255,1 do tcod.console.blit(off1,0,0,80,50,root,0,0) renders the first screen (opaque) tcod.console.blit(off2,0,0,80,50,root,0,0,i/255,i/255) renders the second screen (transparent) tcod.console.flush() end

void **`setKeyboardRepeat`** (int *initialDelay*, int *interval*)

console_offscreen Destroying an offscreen console Use this function to destroy an offscreen console and release any resources allocated.

Don't use it on the root console. TCODConsole::~TCODConsole() void TCOD_console_delete(TCOD_console_t con) console_delete(con) # void TCODConsole::Dispose() through garbage collector con in the C and Python versions, the offscreen console handler *TCODConsole* *off1 = new *TCODConsole(80,50)*; … use off1 delete off1; // destroy the offscreen console TCOD_console_t off1 = TCOD_console_new(80,50); … use off1 TCOD_console_delete(off1); // destroy the offscreen console off1 = libtcod.console_new(80,50) … use off1 libtcod.console_delete(off1) # destroy the offscreen console off1 = tcod.Console(80,50) … use off1 off1=nil release the reference

Public Static Attributes

TCODConsole ***`root`** = new *TCODConsole*()

console_init Initializing the console console

CHAPTER

THREE

SYSTEM LAYER

3.1 Time functions

uint32_t **`TCOD_sys_elapsed_milli`** (void)

float **`TCOD_sys_elapsed_seconds`** (void)

void **`TCOD_sys_sleep_milli`** (uint32_t *val*)

void **`TCOD_sys_set_fps`** (int *val*)

int **`TCOD_sys_get_fps`** (void)

float **`TCOD_sys_get_last_frame_length`** (void)

3.2 Easy screenshots

void **`TCOD_sys_save_screenshot`** (**`const`** char **filename*)

3.3 Miscellaneous utilities

`struct` SDL_Window ***`TCOD_sys_get_sdl_window`** (void)

Return an SDL_Window pointer if one is in use, returns NULL otherwise.

New in version 1.11.

`struct` SDL_Renderer ***`TCOD_sys_get_sdl_renderer`** (void)

Return an SDL_Renderer pointer if one is in use, returns NULL otherwise.

New in version 1.11.

3.4 C++

`class TCODSystem`

Public Static Functions

void **setFps** (int *val*)

system_time system High precision time functions These are functions specifically aimed at real time game development.

Limit the frames per second The setFps function allows you to limit the number of frames per second. If a frame is rendered faster than expected, the TCOD_console_flush function will wait so that the frame rate never exceed this value. You can call this function during your game initialization. You can dynamically change the frame rate. Just call this function once again. **You should always limit the frame rate, except during benchmarks, else your game will use 100% of the CPU power** static void *TCODSystem::setFps(int val)* void TCOD_sys_set_fps(int val) sys_set_fps(val) # static void *TCODSystem::setFps(int val)* tcod.system.setFps(val) val Maximum number of frames per second. 0 means unlimited frame rate.

int **getFps** ()

system_time Get the number of frames rendered during the last second The value returned by this function is updated every second.

static int *TCODSystem::getFps()* int *TCOD_sys_get_fps()* sys_get_fps() # static int *TCODSystem::getFps()* tcod.system.getFps()

float **getLastFrameLength** ()

system_time Get the duration of the last frame This function returns the length in seconds of the last rendered frame.

You can use this value to update every time dependent object in the world. static float *TCODSystem::getLastFrameLength()* float *TCOD_sys_get_last_frame_length()* sys_get_last_frame_length() # static float *TCODSystem::getLastFrameLength()* tcod.system.getLastFrameLength() moving an objet at 5 console cells per second float x=0,y=0; // object coordinates x += 5 * *TCODSystem::getLastFrameLength()*; *TCODConsole::root*->putChar((int)(x),(int)(y),'X'); float x=0,y=0; x += 5 * *TCOD_sys_get_last_frame_length()*; TCOD_console_put_char(NULL,(int)(x),(int)(y),'X'); x=0.0 y=0.0 x += 5 * libtcod.sys_get_last_frame_length() libtcod.console_put_char(0,int(x),int(y),'X') moving an objet at 5 console cells per second x=0 y=0 object coordinates x = x + 5 * tcod.system.getLastFrameLength() libtcod.TCODConsole_root:putChar(x,y,'X')

void **sleepMilli** (uint32_t *val*)

system_time Pause the program Use this function to stop the program execution for a specified number of milliseconds.

static void *TCODSystem::sleepMilli(uint32_t val)* void TCOD_sys_sleep_milli(uint32_t val) sys_sleep_milli(val) # static void TCODSystem::sleepMilli(uint val) tcod.system.sleepMilli(val) val number of milliseconds before the function returns

uint32_t **getElapsedMilli** ()

system_time Get global timer in milliseconds This function returns the number of milliseconds since the program has started.

static uint32_t *TCODSystem::getElapsedMilli()* uint32_t *TCOD_sys_elapsed_milli()* sys_elapsed_milli() # static uint *TCODSystem::getElapsedMilli()* tcod.system.getElapsedMilli()

float **getElapsedSeconds** ()

system_time Get global timer in seconds This function returns the number of seconds since the program has started.

static float *TCODSystem::getElapsedSeconds()* float *TCOD_sys_elapsed_seconds()* sys_elapsed_seconds() # static float *TCODSystem::getElapsedSeconds()* tcod.system.getElapsedSeconds()

TCOD_event_t **waitForEvent** (int *eventMask*, *TCOD_key_t *key*, *TCOD_mouse_t *mouse*, bool *flush*)

console_blocking_input Waiting for any event (mouse or keyboard) This function waits for an event from the user.

The eventMask shows what events we're waiting for. The return value indicate what event was actually triggered. Values in key and mouse structures are updated accordingly. If flush is false, the function waits only if there are no pending events, else it returns the first event in the buffer. typedef enum { TCOD_EVENT_NONE=0, TCOD_EVENT_KEY_PRESS=1, TCOD_EVENT_KEY_RELEASE=2, TCOD_EVENT_KEY=TCOD_EVENT_KEY_PRESS|TCOD_EVENT_KEY_RELEASE, TCOD_EVENT_MOUSE_MOVE=4, TCOD_EVENT_MOUSE_PRESS=8, TCOD_EVENT_MOUSE_RELEASE=16, TCOD_EVENT_MOUSE=TCOD_EVENT_MOUSE_MOVE|TCOD_EVENT_ TCOD_EVENT_ANY=TCOD_EVENT_KEY|TCOD_EVENT_MOUSE, } TCOD_event_t; static TCOD_event_t *TCODSystem::waitForEvent(int eventMask, TCOD_key_t *key, TCOD_mouse_t *mouse, bool flush)* TCOD_event_t TCOD_sys_wait_for_event(int eventMask, TCOD_key_t *key, TCOD_mouse_t *mouse, bool flush) sys_wait_for_event(eventMask,key,mouse,flush) eventMask event types to wait for (other types are discarded) key updated in case of a key event. Can be null if eventMask contains no key event type mouse updated in case of a mouse event. Can be null if eventMask contains no mouse event type flush if true, all pending events are flushed from the buffer. Else, return the first available event *TCOD_key_t* key; *TCOD_mouse_t* mouse; TCOD_event_t ev = TCODSystem::waitForEvent(TCOD_EVENT_ANY,&key,&mouse,true); if (ev == TCOD_EVENT_KEY_PRESS && key.c == 'i') { … open inventory … } *TCOD_key_t* key; *TCOD_mouse_t* mouse; TCOD_event_t ev = TCOD_sys_wait_for_event(TCOD_EVENT_ANY,&key,&mouse,true); if (ev == TCOD_EVENT_KEY_PRESS && key.c == 'i') { … open inventory … }

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

console_non_blocking_input Checking for any event (mouse or keyboard) This function checks if an event from the user is in the buffer.

The eventMask shows what events we're waiting for. The return value indicate what event was actually found. Values in key and mouse structures are updated accordingly. typedef enum { TCOD_EVENT_KEY_PRESS=1, TCOD_EVENT_KEY_RELEASE=2, TCOD_EVENT_KEY=TCOD_EVENT_KEY_PRESS|TCOD_EVENT_KEY_RELEASE, TCOD_EVENT_MOUSE_MOVE=4, TCOD_EVENT_MOUSE_PRESS=8, TCOD_EVENT_MOUSE_RELEASE=16, TCOD_EVENT_MOUSE=TCOD_EVENT_MOUSE_MOVE|TCOD_EVENT_ TCOD_EVENT_ANY=TCOD_EVENT_KEY|TCOD_EVENT_MOUSE, } TCOD_event_t; static TCOD_event_t *TCODSystem::checkForEvent(int eventMask, TCOD_key_t *key, TCOD_mouse_t *mouse)* TCOD_event_t TCOD_sys_check_for_event(int eventMask, TCOD_key_t *key, TCOD_mouse_t *mouse) sys_check_for_event(eventMask,key,mouse) eventMask event types to wait for (other types are discarded) key updated in case of a key event. Can be null if eventMask contains no key event type mouse updated in case of a mouse event. Can be null if eventMask contains no mouse event type *TCOD_key_t* key; *TCOD_mouse_t* mouse; TCOD_event_t ev = TCODSystem::checkForEvent(TCOD_EVENT_ANY,&key,&mouse); if (ev == TCOD_EVENT_KEY_PRESS && key.c == 'i') { … open inventory … } *TCOD_key_t* key; *TCOD_mouse_t* mouse; TCOD_event_t ev = TCOD_sys_check_for_event(TCOD_EVENT_ANY,&key,&mouse); if (ev == TCOD_EVENT_KEY_PRESS && key.c == 'i') { … open inventory … }

void **saveScreenshot** (**const** char **filename*)

system_screenshots system Easy screenshots This function allows you to save the current game screen in a png file, or possibly a bmp file if you provide a filename ending with .bmp.

static void *TCODSystem::saveScreenshot(const char *filename)* void TCOD_sys_save_screenshot(const char *filename) sys_save_screenshot(filename) # static void TCODSystem::saveScreenshot(string filename); tcod.system.saveScreenshot(filename) filename Name of the file. If NULL, a filename is automatically generated with the form "./screenshotNNN.png", NNN being the first free number (if a file named screenshot000.png already exist, screenshot001.png will be used, and so on…).

bool **createDirectory**(**const** char **path*)

system_filesystem system Filesystem utilities Those are a few function that cannot be easily implemented in a portable way in C/C++.

They have no Python wrapper since Python provides its own builtin functions. All those functions return false if an error occurred. Create a directory static bool *TCODSystem::createDirectory(const char *path)* bool TCOD_sys_create_directory(const char *path) path Directory path. The immediate father directory (<path>/..) must exist and be writable.

bool **deleteDirectory**(**const** char **path*)

system_filesystem Delete an empty directory static bool *TCODSystem::deleteDirectory(const char *path)* bool TCOD_sys_delete_directory(const char *path) path directory path.

This directory must exist, be writable and empty

bool **deleteFile**(**const** char **path*)

system_filesystem Delete a file static bool *TCODSystem::deleteFile(const char *path)* bool TCOD_sys_delete_file(const char *path) path File path.

This file must exist and be writable.

bool **isDirectory**(**const** char **path*)

system_filesystem Check if a path is a directory static bool *TCODSystem::isDirectory(const char *path)* bool TCOD_sys_is_directory(const char *path) path a path to check

TCOD_list_t **getDirectoryContent**(**const** char **path*, **const** char **pattern*)

system_filesystem List files in a directory To get the list of entries in a directory (including sub-directories, except .

and ..). The returned list is allocated by the function and must be deleted by you. All the const char * inside must be also freed with TCODList::clearAndDelete. static TCODList *TCODSystem::getDirectoryContent(const char *path, const char *pattern)* TCOD_list_t TCOD_sys_get_directory_content(const char *path) path a directory pattern If NULL or empty, returns all directory entries. Else returns only entries matching the pattern. The pattern is NOT a regular expression. It can only handle one '*' wildcard. Examples : *.png, saveGame*, font*.png

bool **fileExists**(**const** char **filename*, ...)

system_filesystem Check if a given file exists In order to check whether a given file exists in the filesystem.

Useful for detecting errors caused by missing files. static bool *TCODSystem::fileExists*(const char *filename, ...) bool TCOD_sys_file_exists(const char * filename, ...) filename the file name, using printf-like formatting ... optional arguments for filename formatting if (!TCODSystem::fileExists("myfile.%s","txt")) { fprintf(stderr,"no such file!"); } if (!TCOD_sys_file_exists("myfile.%s","txt")) { fprintf(stderr,"no such file!"); }

bool **readFile**(**const** char **filename*, unsigned char ***buf*, size_t **size*)

system_filesystem Read the content of a file into memory This is a portable function to read the content of a file from disk or from the application apk (android).

buf must be freed with free(buf). static bool TCODSystem::readFile(const char *filename, unsigned char **buf, uint32_t *size) bool TCOD_sys_read_file(const char *filename, unsigned char **buf, uint32_t *size) filename the file name buf a buffer to be allocated and filled with the file content size the size of the allocated buffer. unsigned char *buf; uint32_t size; if (*TCODSystem::readFile*("myfile.dat",&buf,&size)) { do something with buf free(buf); } if (TCOD_sys_read_file("myfile.dat",&buf,&size)) { do something with buf free(buf); }

bool **writeFile**(**const** char **filename*, unsigned char **buf*, uint32_t *size*)

system_filesystem Write the content of a memory buffer to a file This is a portable function to write some data to a file.

static bool *TCODSystem::writeFile(const char *filename, unsigned char *buf, uint32_t size)* bool TCOD_sys_write_file(const char *filename, unsigned char *buf, uint32_t size) filename the file name buf a buffer containing the data to write size the number of bytes to write. *TCODSystem::writeFile*("myfile.dat",buf,size)); TCOD_sys_write_file("myfile.dat",buf,size));

void **registerSDLRenderer**(ITCODSDLRenderer **renderer*)

system_sdlcbk system Draw custom graphics on top of the root console You can register a callback that will be called after the libtcod rendering phase, but before the screen buffer is swapped.

This callback receives the screen SDL_Surface reference. This makes it possible to use any SDL drawing functions (including openGL) on top of the libtcod console. Render custom graphics To disable the custom renderer, call the same method with a NULL parameter. Note that to keep libtcod from requiring the SDL headers, the callback parameter is a void pointer. You have to include SDL headers and cast it to SDL_Surface in your code. class TCODLIB_API ITCODSDLRenderer { public : virtual void render(void *sdlSurface) = 0; }; static void *TCODSystem::registerSDLRenderer(ITCODSDLRenderer *callback)*; typedef void (*SDL_renderer_t) (void *sdl_surface); void TCOD_sys_register_SDL_renderer(SDL_renderer_t callback) def renderer (sdl_surface) : … TCOD_sys_register_SDL_renderer(callback) callback The renderer to call before swapping the screen buffer. If NULL, custom rendering is disabled class MyRenderer : public ITCODSDLRenderer { public : void render(void *sdlSurface) { SDL_Surface *s = (SDL_Surface *)sdlSurface; … draw something on s } }; TCODSystem::registerSDLRenderer(new MyRenderer()); void my_renderer(void *sdl_surface) { SDL_Surface *s = (SDL_Surface *)sdl_surface; … draw something on s } TCOD_sys_register_SDL_renderer(my_renderer); def my_renderer(sdl_surface) : … draw something on sdl_surface using pygame libtcod.sys_register_SDL_renderer(my_renderer)

void **forceFullscreenResolution**(int *width*, int *height*)

system_sdlcbk Managing screen redraw libtcod is not aware of the part of the screen your SDL renderer has updated.

If no change occurred in the console, it won't redraw them except if you tell him to do so with this function void TCODConsole::setDirty(int x, int y, int w, int h) void TCOD_console_set_dirty(int x, int y, int w, int h) TCOD_console_set_dirty(x, y, w, h) x,y,w,h Part of the root console you want to redraw even if nothing has changed in the console back/fore/char. system_misc system Miscellaneous utilities Using a custom resolution for the fullscreen mode This function allows you to force the use of a specific resolution in fullscreen mode. The default resolution depends on the root console size and the font character size. static void *TCODSystem::forceFullscreenResolution(int width, int height)* void TCOD_sys_force_fullscreen_resolution(int width, int height) sys_force_fullscreen_resolution(width, height) # static void *TCODSystem::forceFullscreenResolution(int width, int height)*; tcod.system.forceFullscreenResolution(width,height) width,height Resolution to use when switching to fullscreen. Will use the smallest available resolution so that : resolution width >= width and resolution width >= root console width * font char width resolution width >= height and resolution height >= root console height * font char height TCODSystem::forceFullscreenResolution(800,600); // use 800x600 in fullscreen instead of 640x400 *TCODConsole::initRoot*(80,50,"",true); // 80x50 console with 8x8 char => 640x400 default resolution TCOD_sys_force_fullscreen_resolution(800,600); TCOD_console_init_root(80,50,"",true); libtcod.sys_force_fullscreen_resolution(800,600) libtcod.console_init_root(80,50,"",True) tcod.system.forceFullscreenResolution(800,600) use 800x600 in fullscreen instead of 640x400 tcod.console.initRoot(80,50,"",true) 80x50 console with 8x8 char => 640x400 default resolution

void **getCurrentResolution**(int **w*, int **h*)

system_misc Get current resolution You can get the current screen resolution with getCurrentResolution.

You can use it for example to get the desktop resolution before initializing the root console. static void *TCODSystem::getCurrentResolution(int *width, int *height)* void TCOD_sys_get_current_resolution(int *width, int *height) sys_get_current_resolution() # returns w,h # static void TCODSystem::getCurrentResolution(out int w, out int h); width,height contains current resolution when the function returns

void **getFullscreenOffsets**(int **offx*, int **offy*)

system_misc Get fullscreen offset If the fullscreen resolution does not matches the console size in pixels, black borders are added.

This function returns the position in pixels of the console top left corner in the screen. static void *TCODSystem::getFullscreenOffsets(int *offx, int *offy)* void TCOD_sys_get_fullscreen_offsets(int *offx, int *offy) # static void TCODSystem::getFullscreenOffsets(out int offx, out int offy); offx,offy contains the position of the console on the screen when using fullscreen mode.

void **getCharSize**(int **w*, int **h*)

system_misc Get the font size You can get the size of the characters in the font static void *TCODSystem::getCharSize(int *width, int *height)* void TCOD_sys_get_char_size(int *width, int *height) sys_get_char_size() # returns w,h # static void TCODSystem::getCharSize(out int w, out int h); width,height contains a character size when the function returns

void **updateChar**(int *asciiCode*, int *fontx*, int *fonty*, **const** TCODImage **img*, int *x*, int *y*)

system_misc Dynamically updating the font bitmap You can dynamically change the bitmap of a character in the font.

All cells using this ascii code will be updated at next flush call. static void *TCODSystem::updateChar(int asciiCode, int fontx, int fonty,const TCODImage *img,int x,int y)* void TCOD_sys_update_char(int asciiCode, int fontx, int fonty, TCOD_image_t img, int x, int y) sys_update_char(asciiCode,fontx,fonty,img,x,y) asciiCode ascii code corresponding to the character to update fontx,fonty coordinate of the character in the bitmap font (in characters, not pixels) img image containing the new character bitmap x,y position in pixels of the top-left corner of the character in the image

void **setRenderer**(*TCOD_renderer_t renderer*)

system_misc Dynamically change libtcod's internal renderer As of 1.5.1, libtcod contains 3 different renderers : SDL : historic libtcod renderer.

Should work and be pretty fast everywhere OpenGL : requires OpenGL compatible video card. Might be much faster or much slower than SDL, depending on the drivers GLSDL : requires OpenGL 1.4 compatible video card with GL_ARB_shader_objects extension. Blazing fast if you have the proper hardware and drivers. This function switches the current renderer dynamically. static void *TCODSystem::setRenderer(TCOD_renderer_t renderer)* void TCOD_sys_set_renderer(TCOD_renderer_t renderer) sys_set_renderer(renderer) # static void TCODSystem::setRenderer(TCODRendererType renderer); renderer Either TCOD_RENDERER_GLSL, TCOD_RENDERER_OPENGL or TCOD_RENDERER_SDL

TCOD_renderer_t **getRenderer**()

system_misc Get the current internal renderer static TCOD_renderer_t *TCODSystem::getRenderer()* TCOD_renderer_t TCOD_sys_get_renderer() sys_get_renderer() # static TCODRendererType *TCODSystem::getRenderer()*;

bool **setClipboard**(**const** char **value*)

system_clipboard Clipboard integration With these functions, you can copy data in your operating system's clipboard from the game or retrieve data from the clipboard.

system Set current clipboard contents Takes UTF-8 text and copies it into the system clipboard. On Linux, because an application cannot access the system clipboard unless a window is open, if no window is open the call will do nothing. static bool *TCODSystem::setClipboard(const char *value)* bool TCOD_sys_clipboard_set(const char *value) sys_clipboard_set(value) value UTF-8 text to copy into the clipboard

char ***getClipboard**()

system_clipboard Get current clipboard contents Returns the UTF-8 text currently in the system clipboard.

On Linux, because an application cannot access the system clipboard unless a window is open, if no window is open an empty string will be returned. For C and C++, note that the pointer is borrowed, and libtcod will take care of freeing the memory. static char *TCODSystem::getClipboard() char *TCOD_sys_clipboard_get() sys_clipboard_get() # Returns UTF-8 string

CHAPTER

FOUR

ALL PURPOSE CONTAINER

CHAPTER FIVE

COMPRESSION TOOLKIT

CHAPTER

SIX

FILE PARSER

CHAPTER

SEVEN

IMAGE TOOLKIT

CHAPTER

EIGHT

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

NINE

MOUSE SUPPORT

CHAPTER

TEN

NOISE GENERATOR

CHAPTER

ELEVEN

PSUEDORANDOM NUMBER GENERATOR

CHAPTER

TWELVE

BSP

CHAPTER

THIRTEEN

FIELD OF VIEW

CHAPTER

FOURTEEN

HEIGHTMAP TOOLKIT

CHAPTER

FIFTEEN

NAME GENERATOR

CHAPTER

SIXTEEN

PATHFINDING

CHAPTER SEVENTEEN

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++

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

Listing 2: Python

```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