The CODL library documentation

Maxim “Celtrecium”

April 19, 2021

Contents

1 Introduction

The CODL library is a library that allows you to create applications using console graphics. It was inspired by ncurses-based applications.

At the moment this library is very simple and contains basic functions. In the future, it is planned to create an add-on with the ability to create an user interface.

2 Basic functions and concepts

The main role in the library is played “windows” — these are buffers that have a certain position on the screen, width, height, layer, and such internal settings as the background color, symbol color, cursor position in the buffer, alpha channel, and window visibility.

2.1 Initialize, end program

2.1.1 codl_initialize

`int codl_initialize(void)`

Required to start working with the library.
Initializes two screen buffers and a terminal window, clears the contents of the terminal emulator.

2.1.2 codl_end

`int codl_end(void)`

Required to end working with the library.
Clears all memory allocated for window buffers.

For example:

```c
#include <codl.h>

int
main (void)
{
  codl_initialize ();
  ...
  codl_end ();

  return EXIT_SUCCESS;
}
```

2.2 Window type

`codl_window` is a structure that includes the window buffer and its parameters:

struct codl_window *parent_win

Pointer on the parent window (By default, it refers to `term_window`)

int x_position, int y_position

Absolute location of the window on the X, Y axis

int ref_x_position, int ref_y_position

Location of the window on the X, Y axis, relative to the parent window

int width, int height

Size of the window

int layer

Window layer. For example: The window on layer 1 will be below the window on layer 2.

int cursor_pos_x, int cursor_pos_y

Buffer cursor location on the X, Y axis

int s_cur_pos_x, int s_cur_pos_y

Saved cursor buffer location (needed for save and restore cursor position)

int colour_bg, int colour_fg

Buffer background and foreground colors

char alpha

Alpha channel setting (`CODL_ENABLE` or `CODL_DISABLE`) If this attribute is activated, the empty space of the window will not be displayed on the overall composition.

char text_attribute

Text attribute setting (`CODL_BOLD, CODL_ITALIC, CODL_UNDERLINE, CODL_CROSSED_OUT, CODL_DIM`)

int window_visible

Window visibility setting (`CODL_ENABLE` or `CODL_DISABLE`)

char ***window_buffer

Window buffer

2.3 Create and destroy window

2.3.1 codl_create_window

`codl_window *codl_create_window(codl_window *p_win, int layer, int x_pos, int y_pos, int width, int height)`

Parameters:

- `*p_win` — parent window pointer (If the pointer is `NULL`, the parent window is set as term_window)
- `layer` — layer of the window being created
- `x_pos, y_pos` — location of the window on the X, Y axis, relative to the parent window
- `width, heigth` — size of the window being created

2.3.2 codl_destroy_window

`int codl_destroy_window(codl_window *win)`

This function has only one argument — a pointer to the window to destroy

For example:

```c
#include <codl.h>

int
main (void)
{
  /* We can't create a window until the library is initialized */
  codl_window *first_win = NULL;
  codl_window *second_window = NULL;

  codl_initialize ();

  /* We create the first window with next parameters:
   * Parent window                = NULL (Refers to the term_window)
   * Layer of the window          = 1
   * Rel. position on the X axis = 5
   * Rel. position on the Y axis = 5
   * Width                        = 20
   * Height                       = 10
   */
  first_window = codl_create_window (NULL, 1, 5, 5, 20, 10);

  /* We are also creating a second window. Now its parent
     window will be our first window
   */
  second_window = codl_create_window (first_window, 2, 2, 10, 5);

  /*
   *                      Some code
   */

  /* If you need to get rid of one of the windows,
   * you can call the code_destroy_window function.
   *
   * For example:
   */

  codl_destroy_window (second_window);

  /* Now our second window has been deleted and the pointer to it is NULL */

  /*
   *                    Some more code
   */

  codl_end ();

  return EXIT_SUCCESS;
}
```

2.4 Image type

The `cool_image` type partially repeats the code_window type, except that it has only two attributes: width and height

- `int width`
- `int height`
- `char ***image_buffer`

This buffer is needed to repeatedly load the image from it, so as not to read the image directly from the disk each time.

2.4.1 codl_image_to_window

`int codl_image_to_window(codl_window *win, codl_image *img, int x_pos, int y_pos, int x_reg, int y_reg, int width, int height)`

This function transfers the area of image (which is selected by the parameters x_reg, y_reg, width, height) from codl_image to the window buffer at coordinates X, Y.

2.4.2 codl_save_buffer_to_file

`int codl_save_buffer_to_file(codl_window *win, const char *filename)`

In the first argument, we specify the window whose buffer we want to save. In the second argument, we specify the name of the file in which the buffer will be saved.

2.4.3 codl_load_buffer_from_file

`int codl_load_buffer_from_file(codl_window *win, const char *filename, int x_pos, int y_pos)`

This function loads an image from a file directly into the window buffer at X, Y coordinates.

2.4.4 codl_load_image

`codl_image *codl_load_image(const char *filename)`

This function loads an image from a file into the codl_image buffer

3 Library error system

The library error system is implemented quite simply: if a library function fails, it returns a null value and uses the `codl_set_fault` function to determine the cause of this error.

You can get the `CODL_FAULTS` enum value of this error using `code_get_fault_enum` or get a string explaining the error using `codl_get_fault_string`.

4 Setter functions

This section lists all the functions for manipulating windows with a brief description of them.

4.1 Color setters

The first argument in these functions is the window to which the property is applied.

4.1.1 codl_set_colour

`int codl_set_colour(codl_window *win, int bg, int fg)`

This function takes three parameters:

- A pointer to a window
- Background color (0 to 256)
- Foreground color (0 to 256)

4.2 Text attribute setters

The following attributes are available for attribute setters:

1. `CODL_NO_ATTRIBUTES` — zero attribute
2. `CODL_BOLD` — makes the text bold
3. `CODL_ITALIC` — makes the text italicized
4. `CODL_UNDERLINE` — makes the text underlined
5. `CODL_CROSSED_OUT` — makes the text crossed out
6. `CODL_DIM` — makes the text dim

You can also combine these attributes by using a logical OR (`||`)

For example:

```
codl_set_attribute (window_name, CODL_BOLD || CODL_ITALIC || CODL_UNDERLINE);
```

4.2.1 codl_set_attribute

```
int codl_set_attribute(codl_window *win, char attribute)
```

This function sets the window text attributes completely.

4.2.2 codl_add_attribute

```
int codl_add_attribute(codl_window *win, char attribute)
```

This function adds a text attributes to the already set ones.

4.2.3 codl_remove_attribute

```
int codl_remove_attribute(codl_window *win, char attribute)
```

This function deletes the attributes specified in the argument.

4.3 Window attribute setters

4.3.1 codl_set_alpha

```
int codl_set_alpha(codl_window *win, CODL_SWITCH alpha);
```

This function enables or disables the alpha mode of the window.

The first argument in this function is the window to which the property is applied. The second argument can take two values: `CODL_ENABLE` or `CODL_DISABLE`

4.3.2 codl_set_window_visibility

```
int codl_set_window_visible(codl_window *win, CODL_SWITCH visible)
```

This function enables or disables window visibility.

The second argument can take two values: `CODL_ENABLE` or `CODL_DISABLE`

4.3.3 codl_set_cursor_position

```
int codl_set_cursor_position(codl_window *win, int x_pos, int y_pos)
```

This function sets the position of the cursor in the buffer by X, Y coordinates. If the horizontal position overflows, the buffer is shifted down. If the vertical position overflows, the cursor moves to the next line.

4.3.4 codl_save_cursor_position

```
int codl_save_cursor_position(codl_window *win)
```

This function saves cursor position in the window to a special field it the `codl_window` structure.

4.3.5 codl_restore_cursor_position

`int codl_restore_cursor_position(codl_window *win)`

This function restores cursor position from `s_cur_pos_*` fields of the `codl_window` structure.

4.3.6 codl_resize_window

`int codl_resize_window(codl_window *win, int width, int height)`

This function sets the size of the window (width, length).

4.3.7 codl_set_window_position

`int codl_set_window_position(codl_window *win, int new_x_pos, int new_y_pos)`

This function sets the position of the window in X, Y coordinates relative to its parent window.

4.3.8 codl_set_layer

`int codl_set_layer(codl_window *win, int layer)`

This function sets the layer on which the window

4.3.9 codl_window_clear

`int codl_window_clear(codl_window *win)`

This function clears the window buffer.

4.4 Terminal attribute setters

It is not recommended to use these functions while working with the library (except for codl_clear(if this function is followed by codl_redraw or the program terminates) and codl_monochrome_mode)

4.4.1 codl_cursor_mode

`void codl_cursor_mode(CODL_CURSOR cur)`

This function sets the terminal cursor mode: `CODL_SHOW` or `CODL_HIDE`

4.4.2 codl_echo

`int codl_echo(void)`

Enables echo mode (when keyboard input is displayed on stdout).

4.4.3 codl_noecho

`int codl_noecho(void)`

Disables echo mode.

4.4.4 codl_monochrome_mode

`void codl_monochrome_mode(CODL_SWITCH mode)`

Enables monochrome mode (text does not have colors and attributes set).

4.4.5 codl_clear

`void codl_clear(void)`

Clears the terminal screen (not terminal window)

4.5 Primitive setters (Frame setters)

Frame setters work like this: they set parameters for drawing a frame like this

```
4 ----- 2 ----- 5
|               |
|               |
0               1
|               |
|               |
6 ----- 3 ----- 7
```

In function arguments, these parts of the frame can be denoted by a prefix (`fg_` or `ch_`) and the number of this part.

4.5.1 codl_set_frame_colours

`int codl_set_frame_colours(int fg_0, int fg_1, int fg_2, int fg_3, int fg_4, int fg_5, int fg_6, int fg_7)`

Sets the colors for drawing the frame (to understand the arguments, follow the instructions above)

4.5.2 codl_set_frame_symbols

`int codl_set_frame_symbols(char *ch_0, char *ch_1, char *ch_2, char *ch_3, char *ch_4, char *ch_5, char *ch_6, char *ch_7)`

Sets the characters that the frame will be drawn with. Also in the library there are preset symbols for drawing a frame.

4.6 Error system setters

4.6.1 codl_set_fault

`int codl_set_fault(CODL_FAULTS fault_en, const char *fault_str)`

This function sets the error value to the internal library buffer.
The first argument is a CODL_FAULTS enum value:

- `CODL_MEMORY_ALLOCATION_FAULT` — error occurs when allocating memory
- `CODL_NULL_POINTER` — error occurs when pointer is `NULL` value
- `CODL_INVALID_SIZE` — error occurs when the size is not suitable
- `CODL_NOT_INITIALIZED` — error occurs when the library is not initialized

The second argument is a string with an explanation of the error.

4.7 Tab width setter

By default tab width equals 8 spaces

4.7.1 codl_set_tab_width

`void codl_set_tab_width(int width)`

Sets a tab width.

4.8 Image setters

4.8.1 codl_clear_image

`int codl_clear_image(codl_image *img)`

Clears the `codl_image` buffer

5 Getter functions

5.1 Window getters

5.1.1 codl_get_num_of_wins

`int codl_get_num_of_wins(void)`

This function returns the number of windows

5.1.2 codl_get_term

`codl_window *codl_get_term(void)`

This function returns a pointer to the term_window window. This window can be used for drawing, writing. You can also use it to find out the size of the terminal screen.

5.2 Terminal getters

5.2.1 codl_get_term_size

`int codl_get_term_size(int *width, int *height)`

This function takes as arguments pointers to variables of the int type, in which the width and height of the terminal screen will be written.

If you want to know the size of the terminal screen, you'd better do it with the code_get_term getter.

For example: `codl_get_term ()->width`, `codl_get_term ()->height`

5.2.2 codl_resize_term

`int codl_resize_term(void)`

This function checks whether the size of the terminal has changed, and if it has changed, sets the new size of the terminal window and returns the value 1.

5.3 Error system getters

5.3.1 codl_get_fault_string

`char *codl_get_fault_string(void)`

This function returns a pointer to a string with an error explanation.

5.3.2 codl_get_fault_enum

`CODL_FAULTS codl_get_fault_enum(void)`

This function returns the CODL_FAULTS enum value.

5.4 Tab width getter

5.4.1 codl_get_tab_width

`int codl_get_tab_width(void)`

This function returns the value of the tab width.

5.5 Input getters

5.5.1 codl_get_key

`unsigned int codl_get_key(void)`

If the key was pressed, this function returns:

- The ASCII value of the key
- The value of the key that is listed in the CODL_KEY enum

- The value of CODL_KEY_UNICODE, in the case of which we can call the getter codl_get_stored_key to get the unicode value of the key

Or 0 value if a key has not been pressed.

5.5.2 codl_get_stored_key

`char *codl_get_stored_key(void)`

This function returns a pointer to the buffer where the unicode key was written (the size of this buffer is 4, because the maximum UTF-8 character size is 4 bytes)

You can also use the `strcmp` function from the standard library to compare a pressed key with a unicode character.

For example:

```c
#include <codl.h>

int
main (void)
{
  unsigned int key = 0;

  codl_initialize ();

  /* The loop will end if the resulting key is equal to the Escape
   * key code (this code can be viewed in CODL_KEY enum)
   */
  while ((key = codl_get_key ()) != CODL_KEY_ESC)
    {
      switch (key)
        {
        case 0:
          continue;

        case CODL_KEY_UP:
          codl_write (codl_get_term (), "Oh, honey, you pushed the up button..."
                                        " Push something else;)\n");

          break;

        case CODL_KEY_UNICODE:
          codl_write (codl_get_term (), "Wow, you hit the button ");
          codl_write (codl_get_term (), codl_get_stored_key ());
          codl_write (codl_get_term (), "\n");

          break;

        default:
          codl_write (codl_get_term (), "You don't spoil me... Can you press "
                                        "the up key or some non-ASCII key?\n");

          break;
        }

      codl_display ();
    }

  codl_end ();

  return EXIT_SUCCESS;
}
```

5.6 String getters

5.6.1 codl_strlen

`size_t codl_strlen(const char *string)`

This function is analog of `strlen` function from `string.h`

5.6.2 codl_string_length

`size_t codl_string_length(const char *string)`

This function counts the number of characters in a string. Supports UTF-8

6 Functions for manipulating the buffer

This section contains functions for working with the `codl_window` buffer.

6.1 codl_buffer_scroll_down

`int codl_buffer_scroll_down(codl_window *win, int down)`

This function shifts the contents of the window buffer by a certain number of characters down.

6.2 codl_buffer_scroll_up

`int codl_buffer_scroll_up(codl_window *win, int down)`

This function shifts the contents of the window buffer by a certain number of characters up.

7 Functions for writing and drawing primitives

The functions of this section output text to the terminal with the attributes that you set with the color and text setters (except for the codl_frame function, which has its own setters (the frame takes the background color from the window attributes))

7.1 Functions for writing

7.1.1 codl_write

`int codl_write(codl_window *win, char *string)`

This function writes a string to the window buffer. It is the main function of writing to a window buffer. Supports parsing ANSI sequences. For example:

```
codl_write (window_name, "\033[1mHello world!\033[0m");
```

7.1.2 codl_replace_attributes

`int codl_replace_attributes(codl_window *win, int x0_pos, int y0_pos, int x1_pos, int y1_pos)`

This function replaces the text attributes with those that you previously set using color and text setters in the region marked with coordinates x_0, y_0, x_1, y_1

7.2 Functions for drawing primitives

7.2.1 codl_line

`int codl_line(codl_window *win, int x1, int y1, int x2, int y2, char *symbol)`

This function draws a line at coordinates x_0, y_0, x_1, y_1 using a character, which is specified as the last argument using a string literal

7.2.2 codl_rectangle

```
int codl_rectangle(codl_window *win, int x0_pos, int y0_pos, int x1_pos, int y1_pos, char *symbol)
```

This function draws a rectangle at coordinates x_0, y_0, x_1, y_1 using a character, which is specified as the last argument using a string literal

7.2.3 codl_rectangle

```
int codl_rectangle(codl_window *win, int x0_pos, int y0_pos, int x1_pos, int y1_pos, char *symbol)
```

This function draws a frame at coordinates x_0, y_0, x_1, y_1 using a characters set by `codl_set_frame_symbols` function and with colors set by `codl_set_frame_colours` function.

8 Functions for working with memory

8.1 Memory (re-)allocation functions

This subsection contains wrappers over the memory allocation functions from the standard library. These are safe functions that have integration with the error system of this library.

8.1.1 codl_malloc_check

```
void *codl_malloc_check(size_t size)
```

This function allocates `size` bytes on the heap and returns a pointer to the beginning of this area.

8.1.2 codl_realloc_check

```
void *codl_realloc_check(void *ptrmem, size_t size)
```

This function reallocates memory blocks. The size of the memory block referred to by the `ptrmem` parameter is changed to size bytes. The memory block can shrink or grow in size.

8.1.3 codl_calloc_check

```
void *codl_calloc_check(size_t number, size_t size)
```

The calloc function allocates a block of memory for an array of `number` elements, each of which is `size` bytes, and initializes all of its bits to zeros. As a result, a memory block of `number` * `size` bytes is allocated, and the entire block is filled with zeros.

8.2 Set and copy memory functions

This subsection contains the safe counterparts of the standard library functions.

8.2.1 codl_memset

```
int codl_memset(void *dest, codl_rsize_t destsize, int ch, codl_rsize_t count)
```

This function fills the `count` bytes of memory at `dest` with `ch`. If `count` is bigger than `destsize`, the function sets `destsize` bytes of memory.

8.2.2 codl_memcpy

```
int codl_memcpy(void *dest, codl_rsize_t destsize, const void *src, codl_rsize_t count)
```

This function copies `count` bytes of memory from `src` to `dest`. If `count` is greater than `destsize`, the function copies `destsize` bytes of memory. This function is protected from memory overlap.

9 Display functions

9.1 codl_display

`int codl_display(void)`

This function is engaged in displaying the picture and all its changes on the screen of your terminal. This is the main display function, in most cases you need to use it.

9.2 codl_redraw

`int codl_redraw(void)`

This function completely redraws the image on the screen. It may be useful after using the `codl_clear` function, in other cases it is better to refrain from using it.

9.3 codl_redraw_diff

`int codl_redraw_diff(void)`

This function can re-display the changes that have occurred on the screen. The function is needed only in theory, in practice it has not yet been used.

10 Other functions

This section contains features that do not fall into other categories

10.1 codl_itoa

`char *codl_itoa(int num, char *string)`

This function converts an int value to a string.

10.2 codl_input_form

`int codl_input_form(codl_window *win, char **str, int pos_x, int pos_y, size_t size)`

This function creates a form for input in a `win` window of size `size` characters, which will be located at the x, y coordinates relative to the specified window. Also, this function accepts a pointer to a string in order to write the result into it after the end of the work. The memory for the row is allocated on the heap, so remember to clear the memory after you finish.

11 Some more examples

11.1 Image demo

```c
#include <codl.h>

int
main (void)
{
  codl_window *win   = NULL;
  codl_window *s_win = NULL;
  codl_image  *img   = NULL;

  codl_initialize ();
  /* Creating window */
  win = codl_create_window (NULL, 1, 5, 5, 24, 8);

  /* Creating centered child window of "win" */
  s_win = codl_create_window (win, 2, 2, 2, win->width - 4, win->height - 4);

  /* Setting window color attributes for drawing rectangle for fill window
   * buffer with solid color
   */
  codl_set_colour (win, CODL_BRIGHT_GREEN, CODL_DEFAULT_COLOUR);

  /* Fill the window buffer with rectangle */
  codl_rectangle (win, 0, 0, win->width, win->height, " ");

  /* Draw window frame with default frame settings and bright green
   * background color
   */
  codl_frame (win, 0, 0, win->width, win->height);

  /* Setting cursor position, text and colour attributes for writing */
  codl_set_cursor_position (win, 6, 1);
  codl_set_colour (win, CODL_BLUE, CODL_BRIGHT_WHITE);
  codl_set_attribute (win, CODL_BOLD | CODL_UNDERLINE);

  /* Write "Hello world!" */
  codl_write (win, "Hello world!");

  /* Write some text to the second window with default attributes */
  codl_write (s_win, "This is some text in second window :P\nYou wrote: ");

  /* Save the window buffer of "win" to file "file_image.cdl" */
  codl_save_buffer_to_file (win, "file_image.cdl");

  /* Load our file to image buffer "img" */
  img = codl_load_image ("file_image.cdl");

  /* Load image from image buffer to terminal window buffer */
  codl_image_to_window(codl_get_term (), img, 14, 15, 0, 0,
                        img->width, img->height);

  /* Free our pointer after using */
  codl_clear_image (img);

  /* Display our results */
  codl_display ();

  /* As a result, we have a copy of the image from the win window
   * in our terminal window.
   */

  codl_end ();

  return EXIT_SUCCESS;
}
```

11.2 Small demo

```c
#include <codl.h>

int
main (void)
{
  codl_window *win   = NULL;
  codl_window *s_win = NULL;
  char *str          = NULL;
  unsigned int key   = 0;

  codl_initialize ();
  /* Creating window */
  win   = codl_create_window (NULL, 1, 5, 5, 24, 8);

  /* Creating centered child window of "win" */
  s_win = codl_create_window (win, 2, 2, 2, win->width - 4, win->height - 4);

  /* Setting window color attributes for drawing rectangle for fill window
   * buffer with solid color
   */
  codl_set_colour (win, CODL_BRIGHT_GREEN, CODL_DEFAULT_COLOUR);

  /* Fill the window buffer with rectangle */
  codl_rectangle (win, 0, 0, win->width, win->height, " ");

  /* Draw window frame with default frame settings and bright green
   * background color
   */
  codl_frame (win, 0, 0, win->width, win->height);

  /* Setting cursor position, text and colour attributes for writing */
  codl_set_cursor_position (win, 6, 1);
  codl_set_colour (win, CODL_BLUE, CODL_BRIGHT_WHITE);
  codl_set_attribute (win, CODL_BOLD | CODL_UNDERLINE);

  /* Write "Hello world!" */
  codl_write (win, "Hello world!");

  /* Write some text to the second window with default attributes */
  codl_write (s_win, "This is some text in second window :P\nYou wrote: ");

  /* Prompt the user to enter a string */
  codl_set_colour (win, CODL_CYAN, CODL_BRIGHT_WHITE);
  codl_input_form(win, &str, 6, win->height - 2, 11);

  codl_set_attribute (s_win, CODL_BOLD);
  codl_write (s_win, str);

  /* Free memory after using codl_input_form function */
  free (str);

  /* Display our results */
  codl_display ();

  /* Create a loop in which the user can move the main window using the arrows
   */
  while ((key = codl_get_key ()) != CODL_KEY_ESC)
    {
      switch (key) {
      case 0:
        continue;

      case CODL_KEY_RIGHT:
        codl_set_window_position (win, win->x_position + 1, win->y_position);
        break;

      case CODL_KEY_LEFT:
        codl_set_window_position (win, win->x_position - 1, win->y_position);
        break;

      case CODL_KEY_UP:
        codl_set_window_position (win, win->x_position, win->y_position - 1);
        break;

      case CODL_KEY_DOWN:
        codl_set_window_position (win, win->x_position, win->y_position + 1);
        break;
      }

      codl_display ();
    }

  codl_end ();

  return EXIT_SUCCESS;
}
```