
LittlevGL Documentation

Release 6.0

Gabor Kiss-Vamosi

Jun 18, 2019

CONTENTS

1	Introduction	3
1.1	Key features	3
1.2	Requirements	3
2	Get started	5
2.1	Live demos	5
2.2	Micropython	5
2.3	Simulator on PC	5
3	Porting	9
3.1	System overview	9
3.2	Set-up a project	10
3.3	Display interface	10
3.4	Input device interface	13
3.5	Tick interface	16
3.6	Task Handler	16
3.7	Sleep management	17
3.8	Use with an operating system	17
4	Overview	19
4.1	Objects	19
4.2	Layers	23
4.3	Events	24
4.4	Styles	27
4.5	Input devices	33
4.6	Displays	34
4.7	Fonts	34
4.8	Images	39
4.9	File system	39
4.10	Animations	39
4.11	Drawing	40
5	Object types	43
5.1	Base object (lv_obj)	43
5.2	Arc (lv_arc)	43
5.3	Bar (lv_bar)	46
5.4	Button (lv_btn)	51
5.5	Button matrix (lv_btm)	59
5.6	Calendar (lv_calendar)	66
5.7	Canvas (lv_canvas)	73

5.8	Check box (lv_cb)	79
5.9	Chart (lv_chart)	84
5.10	Container	94
5.11	Drop down list (lv_ddlist)	99
5.12	Gauge (lv_gauge)	106
5.13	Image (lv_img)	111
5.14	Image button (lv_imgbtn)	117
5.15	Keyboard (lv_kb)	121
5.16	Label (lv_label)	127
5.17	LED (lv_led)	129
5.18	Line (lv_line)	130
5.19	List (lv_list)	132
5.20	Line meter (lv_lmeter)	135
5.21	Message box (lv_mbox)	138
5.22	Page (lv_page)	138
5.23	Preload (lv_preload)	138
5.24	Roller (lv_roller)	138
5.25	Slider (lv_slider)	138
5.26	Spinbox (lv_spinbox)	140
5.27	Switch (lv_sw)	140
5.28	Table (lv_table)	142
5.29	Tab view (lv_tabview)	142
5.30	Text area (lv_ta)	142
5.31	Tile view (lv_tileview)	142
5.32	Window (lv_win)	142

Index

143

English - Portuguese - Espanol - Turkish



LittlevGL is a free and open-source graphics library providing everything you need to create embedded GUI with easy-to-use graphical elements, beautiful visual effects and low memory footprint.

INTRODUCTION

LittlevGL is a free and open-source graphics library providing everything you need to create embedded GUI with easy-to-use graphical elements, beautiful visual effects and low memory footprint.

1.1 Key features

- Powerful building blocks buttons, charts, lists, sliders, images etc
- Advanced graphics with animations, anti-aliasing, opacity, smooth scrolling
- Various input devices touch pad, mouse, keyboard, encoder etc
- Multi language support with UTF-8 encoding
- Fully customizable graphical elements
- Hardware independent to use with any microcontroller or display
- Scalable to operate with little memory (80 kB Flash, 10 kB RAM)
- OS, External memory and GPU supported but not required
- Single frame buffer operation even with advanced graphical effects
- Written in C for maximal compatibility (C++ compatible)
- Simulator to start embedded GUI design on PC without embedded hardware
- Tutorials, examples, themes for rapid GUI design
- Documentation online and offline
- Free and open-source under MIT licence

1.2 Requirements

- 16, 32 or 64 bit microcontroller or processor
- 16 MHz clock speed
- 8 kB RAM for static data and >2 KB RAM for dynamic data (graphical objects)
- 64 kB program memory (flash)
- Optionally ~1/10 screen sized memory for internal buffering (at 240 × 320, 16 bit colors it means 15 kB)
- C99 or newer compiler

The LittlevGL is designed to be highly portable and to not use any external resources:

- No external RAM required (but supported)
- No float numbers are used
- No GPU needed (but supported)
- Only a single frame buffer is required located in:
 - Internal RAM or
 - External RAM or
 - External display controller's memory

If you would like to reduce the required hardware resources you can:

- Disable the unused object types to save RAM and ROM
- Change the size of the graphical buffer to save RAM (see later)
- Use more simple styles to reduce the rendering time

GET STARTED

- [lvgl on GihHub](#)
- [example projects](#)

2.1 Live demos

See look and feel

2.2 Micropython

play with it in micropython

2.3 Simulator on PC

You can try out the LittlevGL **using only your PC** without any development boards. Write a code, run it on the PC and see the result on the monitor. It is cross-platform: Windows, Linux and OSX are supported. The written code is portable, you can simply copy it when using an embedded hardware.

The simulator is also very useful to report bugs because it means common platform for every user. So it's a good idea the reproduce a bug in simualtor and use the code snippen in the [Forum](#).

2.3.1 Select an IDE

The simulator is ported to valrious IDEs. Choose your favourite IDE, read its README on GitHub, download the project, and load it to the IDE.

In followings the set-up guide of Eclipse CDT is described in more details.

2.3.2 Set-up Eclipse CDT

Install Eclipse CDT

Eclipse CDT is C/C++ IDE. You can use other IDEs as well but in this tutorial the configuration for Eclipse CDT is shown.

Eclipse is a Java based software therefore be sure **Java Runtime Environment** is installed on your system.

On Debian-based distros (e.g. Ubuntu): `sudo apt-get install default-jre`

You can download Eclipse's CDT from: <https://eclipse.org/cdt/>. Start the installer and choose *Eclipse CDT* from the list.

Install SDL 2

The PC simulator uses the [SDL 2](#) cross platform library to simulate a TFT display and a touch pad.

Linux

On **Linux** you can easily install SDL2 using a terminal:

1. Find the current version of SDL2: `apt-cache search libsdl2` (e.g. `libsdl2-2.0-0`)
2. Install SDL2: `sudo apt-get install libsdl2-2.0-0` (replace with the found version)
3. Install SDL2 development package: `sudo apt-get install libsdl2-dev`
4. If build essentials are not installed yet: `sudo apt-get install build-essential`

Windows

If you are using **Windows** firstly you need to install MinGW ([64 bit version](#)). After it do the following steps to add SDL2:

1. Download the development libraries of SDL. Go to <https://www.libsdl.org/download-2.0.php> and download *Development Libraries: SDL2-devel-2.0.5-mingw.tar.gz*
2. Uncompress the file and go to `x86_64-w64-mingw32` directory (for 64 bit MinGW) or to `i686-w64-mingw32` (for 32 bit MinGW)
3. Copy `...mingw32/include/SDL2` folder to `C:/MinGW/.../x86_64-w64-mingw32/include`
4. Copy `...mingw32/lib/` content to `C:/MinGW/.../x86_64-w64-mingw32/lib`
5. Copy `...mingw32/bin/SDL2.dll` to `{eclipse_worksapce}/pc_simulator/Debug/`. Do it later when Eclipse is installed.

Note: If you will use **Microsoft Visual Studio** instead of Eclipse then you don't have to install MinGW.

OSX

On **OSX** you can easily install SDL2 with brew: `brew install sdl2`

If something is not working I suggest [this tutorial](#) to get started with SDL.

Pre-configured project

A pre-configured graphics library project (based on the latest release) is always available. You can find it on [GitHub](#) or on the [Download](#) page. (The project is configured for Eclipse CDT.)

Add the pre-configured project to Eclipse CDT

Run Eclipse CDT. It will show a dialogue about the **workspace path**. Before accepting it check that path and copy (and unzip) the downloaded pre-configured project there. Now you can accept the workspace path. Of course you can modify this path but in that case copy the project to that location.

Close the start up window and go to **File->Import** and choose **General->Existing project into Workspace**. **Browse the root directory** of the project and click **Finish**

On **Windows** you have to do two additional things:

- Copy the **SDL2.dll** into the project's Debug folder
- Right click on the project -> Project properties -> C/C++ Build -> Settings -> Libraries -> Add ... and add *mingw32* above SDLmain and SDL. (The order is important: mingw32, SDLmain, SDL)

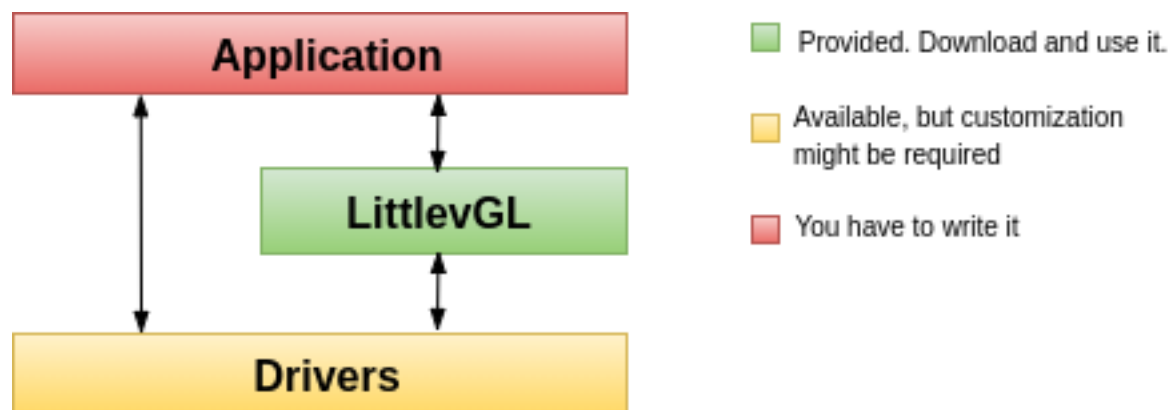
Compile and Run

Now you are ready to run the Littlev Graphics Library on your PC. Click on the Hammer Icon on the top menu bar to Build the project. If you have done everything right you will not get any errors. Note that on some systems additional steps might be required to "see" SDL 2 from Eclipse but in most of cases the configurations in the downloaded project is enough.

After a success build click on the Play button on the top menu bar to run the project. Now a window should appear in the middle of your screen.

Now everything is ready to use the Littlev Graphics Library in the practice or begin the development on your PC.

3.1 System overview



Application Your application which creates the GUI and handles the specific tasks.

LittlevGL The graphics library itself. Your application can communicate with the library to create a GUI. It contains a HAL (Hardware Abstraction Layer) interface to register your display and input device drivers.

Driver Besides your specific drivers, it contains functions to drive your display, optionally to a GPU and to read the touchpad or buttons.

There are **two typical hardware set-ups** depending on the MCU has an LCD/TFT driver periphery or not. In both cases, a frame buffer will be required to store the current image of the screen.

1. **MCU with TFT/LCD driver** If your MCU has a TFT/LCD driver periphery then you can connect a display directly via RGB interface. In this case, the frame buffer can be in the internal RAM (if the MCU has enough RAM) or in the external RAM (if the MCU has a memory interface).
2. **External display controller** If the MCU doesn't have TFT/LCD driver interface then an external display controller (E.g. SSD1963, SSD1306, ILI9341) has to be used. In this case, the MCU can communicate with the display controller via Parallel port, SPI or sometimes I2C. The frame buffer is usually located in the display controller which saves a lot of RAM for the MCU.

3.2 Set-up a project

3.2.1 Get the library

Littlev Graphics Library is available on GitHub: <https://github.com/littlevgl/lvgl>.

You can clone it or download the latest version of the library from GitHub or you can use the [Download](#) page as well.

The graphics library is the **lvgl** directory which should be copied into your project.

3.2.2 Config file

There is a configuration header file for LittlevGL called **lv_conf.h**. It sets the library's basic behavior, disables unused modules and features, adjusts the size of memory buffers in compile time, etc.

Copy **lvgl/lv_conf_template.h** next to the *lvgl* directory and rename it to *lv_conf.h*. Open the file and change the `#if 0` at the beginning to `#if 1` to enable its content.

lv_conf.h can be copied other places as well but then you should add `LV_CONF_INCLUDE_SIMPLE` define to our compilers (e.g. `-DLV_CONF_INCLUDE_SIMPLE` for gcc) and set the include path manually.

In the config file comments explain the meaning of the options. Check at least these three config options and modify them according to your hardware:

1. **LV_HOR_RES_MAX** Your display's horizontal resolution
2. **LV_VER_RES_MAX** Your display's vertical resolution
3. **LV_COLOR_DEPTH** 8 for (RG332), 16 for (RGB565) or 32 for (RGB888 and ARGB8888).

3.2.3 Initialization

In order to use the graphics library you have to initialize it and the other components too. To order of the initialization is:

1. Call *lv_init()*
2. Initialize your drivers
3. Register the display and input devices drivers in LittlevGL. More about *Display* and *Input device* registration.
4. Call *lv_tick_inc(x)* in every x milliseconds in an interrupt to tell the elapsed time. [Learn more](#).
5. Call *lv_task_handler()* periodically in every few milliseconds to handle LittlevGL related tasks. [Learn more](#).

3.3 Display interface

To set up a display an *lv_disp_buf_t* and an *lv_disp_drv_t* variable has to be initialized.

- **lv_disp_buf_t** contains internal graphics buffer(s).
- **lv_disp_drv_t** contains callback functions to interact with the display and manipulate drawing related things.

3.3.1 Display buffer

lv_disp_buf_t can be initialized like this:

```
/*A static or global variable to store the buffers*/
static lv_disp_buf_t disp_buf;

/*Static or global buffer(s). The second buffer is optional*/
static lv_color_t buf_1[MY_DISP_HOR_RES * 10];
static lv_color_t buf_2[MY_DISP_HOR_RES * 10];

/*Inititalize `disp_buf` with the buffer(s) */
lv_disp_buf_init(&disp_buf, buf_1, buf_2, MY_DISP_HOR_RES*10);
```

There are three possible configurations regarding the buffer size:

1. **One buffer** LittlevGL draws the content of the screen into a buffer and sends it to the display. The buffer can be smaller than the screen. In this case, the larger areas will be redrawn in multiple parts. If only small areas changes (e.g. button press) then only those areas will be refreshed.
2. **Two non-screen-sized buffers** having two buffers LittlevGL can draw into one buffer while the content of the other buffer is sent to display in the background. DMA or other hardware should be used to transfer the data to the display to let the CPU draw meanwhile. This way the rendering and refreshing of the display become parallel. Similarly to the *One buffer* LittlevGL will draw the display's content in chunks if the buffer is smaller than the area to refresh.
3. **Two screen-sized buffers.** In contrast to *Two non-screen-sized buffers* LittlevGL will always provide the whole screen's content not only chunks. This way the driver can simply change the address of the frame buffer to the buffer received from LittlevGL. Therefore this method works the best when the MCU has an LCD/TFT interface and the frame buffer is just a location in the RAM.

3.3.2 Display driver

Once the buffer initialization is ready the display drivers need to be initialized. In the most simple case only the following two fields of lv_disp_drv_t needs to be set:

- **buffer** pointer to an initialized lv_disp_buf_t variable.
- **flush_cb** a callback function to copy a buffer's content to a specific area of the display.

There are some optional data fields:

- **hor_res** horizontal resolution of the display. (LV_HOR_RES_MAX by default from lv_conf.h)
- **ver_res** vertical resolution of the display. (LV_VER_RES_MAX by default from lv_conf.h)
- **color_chroma_key** a color which will be drawn as transparent on chrome keyed images. LV_COLOR_TRANSP by default from lv_conf.h)
- **user_data** custom user data for the driver. Its type can be modified in lv_conf.h.
- **antialiasing** use anti-aliasing (edge smoothing). LV_ANTIALIAS by default from lv_conf.h
- **rotated** if 1 swap hor_res and ver_res. LittlevGL draws in the same direction in both cases (in lines from top to bottom) so the driver also needs to be reconfigured to change the display's fill direction.

To use a GPU the following callbacks can be used:

- **mem_fill_cb** fill an area with colors.
- **mem_blend_cb** blend two buffers using opacity.

Some other optional callbacks to make easier and more optimal to work with monochrome, grayscale or other non-standard FGB displays:

- **rounder_cb** round the coordinates of areas to redraw. E.g. a 2x2 px can be converted to 2x8. It can be used if the display controller can refresh only areas with specific height or width (usually 8 px height with monochrome displays).
- **set_px_cb** a custom function to write the *display buffer*. It can be used to store the pixels in a more compact way if the display has a special color format. (e.g. 1 bit monochrome, 2 bit grayscale etc.) This way the buffers used in `lv_disp_buf_t` can be smaller to hold only the required number of bits for the given area size.
- **monitor_cb** a callback function tell how many pixels were refreshed in how much time.

To set the fields of `lv_disp_drv_t` variable it needs to be initialized with `lv_disp_drv_init(&disp_drv)`. And finally to register a display for LittlevGL `lv_disp_drv_register(&disp_drv)` needs to be called.

All together it looks like this:

```
lv_disp_drv_t disp_drv;           /*A variable to hold the drivers. Can be_
↪local variable*/
lv_disp_drv_init(&disp_drv);      /*Basic initialization*/
disp_drv.buffer = &disp_buf;     /*Set an initialized buffer*/
disp_drv.flush_cb = my_flush_cb; /*Set a flush callback to draw to the_
↪display*/
lv_disp_t * disp;
disp = lv_disp_drv_register(&disp_drv); /*Register the dirver and save the_
↪created display objects*/
```

Here some simple examples of the callbacks:

```
void my_flush_cb(lv_disp_drv_t * disp_drv, const lv_area_t * area, lv_color_t * color_
↪p)
{
    /*The most simple case (but also the slowest) to put all pixels to the screen one-
↪by-one*/
    int32_t x, y;
    for(y = area->y1; y <= area->y2; y++) {
        for(x = area->x1; x <= area->x2; x++) {
            put_px(x, y, *color_p)
            color_p++;
        }
    }

    /* IMPORTANT!!!
     * Inform the graphics library that you are ready with the flushing*/
    lv_disp_flush_ready(disp);
}

void my_mem_fill_cb(lv_disp_drv_t * disp_drv, lv_color_t * dest_buf, const lv_area_t_
↪* dest_area, const lv_area_t * fill_area, lv_color_t color);
{
    /*It's an example code which should be done by your GPU*/
    uint32_t x,y;
    for(y = 0; i < length; i++) {
        dest[i] = color;
    }
}

void my_mem_blend_cb(lv_disp_drv_t * disp_drv, lv_color_t * dest, const lv_color_t *_
↪src, uint32_t length, lv_opa_t opa)
```

(continues on next page)

(continued from previous page)

```
{
    /*It's an example code which should be done by your GPU*/
    uint32_t i;
    for(i = 0; i < length; i++) {
        dest[i] = lv_color_mix(dest[i], src[i], opa);
    }
}

void my_rounder_cb(lv_disp_drv_t * disp_drv, lv_area_t * area)
{
    /* Update the areas as needed. Can be only larger.
     * For example to always have lines 8 px height:*/
    area->y1 = area->y1 & 0x07;
    area->y2 = (area->y2 & 0x07) + 8;
}

void my_set_px_cb(lv_disp_drv_t * disp_drv, uint8_t * buf, lv_coord_t buf_w, lv_coord_t x, lv_coord_t y, lv_color_t color, lv_opa_t opa)
{
    /* Write to the buffer as required for the display.
     * Write only 1 bit for monochrome displays mapped vertically:*/
    buf += buf_w * (y >> 3) + x;
    if(lv_color_brightness(color) > 128) (*buf) |= (1 << (y % 8));
    else (*buf) &= ~(1 << (y % 8));
}

void my_monitor_cb(lv_disp_drv_t * disp_drv, uint32_t time, uint32_t px)
{
    printf("%d px refreshed in %d ms\n", time, ms);
}
```

3.3.3 Multi-display support

With LittlevGL multiple displays can be used. Just initialize multiple drivers and buffer and register them. Each display has its own screens and objects on the screens. To get currently active screen of a display use `lv_disp_get_scr_act(disp)` (where `disp` is the return value of `lv_disp_drv_register`). To set a new screen as active on a display use `lv_disp_set_scr_act(screen1)`.

Or in a shorter form set a default display with `lv_disp_set_default(disp)` and get/set the active screen with `lv_scr_act()` and `lv_scr_load()`.

Learn more about screens in the [Screen - the most basic parent](#) section.

3.4 Input device interface

To set up an input device an `lv_indev_drv_t` variable has to be initialized:

```
lv_indev_drv_t indev_drv;
lv_indev_drv_init(&indev_drv);          /*Basic initialization*/
indev_drv.type = ...                    /*See below.*/
indev_drv.read_cb = ...                 /*See below.*/
/*Register the driver in LittlevGL and save the created input device object*/
lv_indev_t * my_indev = lv_indev_drv_register(&indev_drv);
```

type can be

- **LV_INDEV_TYPE_POINTER** touchpad or mouse
- **LV_INDEV_TYPE_KEYPAD** keyboard or keypad
- **LV_INDEV_TYPE_ENCODER** oncoder with left, right, push options
- **LV_INDEV_TYPE_BUTTON** external buttons pressing the screen

read_cb is a function pointer which will be called periodically to report the current state of an input device. It can also buffer data and return `false` when no more data to be read or `true` when the buffer is not empty.

Visit [Input devices](#) to learn more about input devices in general.

3.4.1 Touchpad, mouse or any pointer

Input devices which are able to click points of the screen belong to this category.

```

indev_drv.type = LV_INDEV_TYPE_POINTER;
indev_drv.read_cb = my_input_read;

...

bool my_input_read(lv_indev_drv_t * drv, lv_indev_data_t*data)
{
    data->point.x = touchpad_x;
    data->point.y = touchpad_y;
    data->state = LV_INDEV_STATE_PR or LV_INDEV_STATE_REL;
    return false; /*No buffering now so no more data read*/
}

```

Important: Touchpad drivers must return the last X/Y coordinates even when the state is `LV_INDEV_STATE_REL`.

To set a mouse cursor use `lv_indev_set_cursor(my_indev, &img_cursor)`. (`my_indev` is the return value of `lv_indev_drv_register`)

3.4.2 Keypad or keyboard

Full keyboards with all the letters or simple keypads with a few navigation buttons belong here.

To use a keyboard/keypad:

- Register a `read_cb` function with `LV_INDEV_TYPE_KEYPAD` type.
- Enable `LV_USE_GROUP` in `lv_conf.h`
- An object group has to be created: `lv_group_t * g = lv_group_create()` and objects have to be added to it with `lv_group_add_obj(g, obj)`
- The created group has to be assigned to an input device: `lv_indev_set_group(my_indev, g)` (`my_indev` is the return value of `lv_indev_drv_register`)
- Use `LV_KEY_...` to navigate among the objects in the group. See `lv_core/lv_group.h` for the available keys.

```

indev_drv.type = LV_INDEV_TYPE_KEYPAD;
indev_drv.read_cb = my_input_read;

...

bool keyboard_read(lv_indev_drv_t * drv, lv_indev_data_t*data){
    data->key = last_key();           /*Get the last pressed or released key*/

    if(key_pressed()) data->state = LV_INDEV_STATE_PR;
    else data->state = LV_INDEV_STATE_REL;

    return false; /*No buffering now so no more data read*/
}

```

3.4.3 Encoder

With an encoder you can do 4 things:

1. Press its button
2. Long press its button
3. Turn left
4. Turn right

In short, the Encoder input devices work like this:

- By turning the encoder you can focus on the next/previous object.
- When you press the encoder on a simple object (like a button), it will be clicked.
- If you press the encoder on a complex object (like a list, message box, etc.) the object will go to edit mode where by turning the encoder you can navigate inside the object.
- To leave edit mode press long the button.

To use an *Encoder* (similarly to the *Keypads*) the objects should be added to groups.

```

indev_drv.type = LV_INDEV_TYPE_ENCODER;
indev_drv.read_cb = my_input_read;

...

bool encoder_read(lv_indev_drv_t * drv, lv_indev_data_t*data){
    data->enc_diff = enc_get_new_moves();

    if(enc_pressed()) data->state = LV_INDEV_STATE_PR;
    else data->state = LV_INDEV_STATE_REL;

    return false; /*No buffering now so no more data read*/
}

```

3.4.4 Button

Buttons mean external “hardware” buttons next to the screen which are assigned to specific coordinates of the screen. If a button is pressed it will simulate the pressing on the assigned coordinate. (Similarly to a touchpad)

To assign buttons to coordinates use `lv_indev_set_button_points(my_indev, points_array)`. `points_array` should look like `const lv_point_t points_array[] = { {12, 30}, {60, 90}, ... }`

```
indev_drv.type = LV_INDEV_TYPE_BUTTON;
indev_drv.read_cb = my_input_read;

...

bool button_read(lv_indev_drv_t * drv, lv_indev_data_t*data){
    static uint32_t last_btn = 0;    /*Store the last pressed button*/
    int btn_pr = my_btn_read();      /*Get the ID (0,1,2...) of the pressed button*/
    if(btn_pr >= 0) {                 /*Is there a button press? (E.g. -1 indicated no_
↪button was pressed)*/
        last_btn = btn_pr;           /*Save the ID of the pressed button*/
        data->state = LV_INDEV_STATE_PR; /*Set the pressed state*/
    } else {
        data->state = LV_INDEV_STATE_REL; /*Set the released state*/
    }

    data->btn = last_btn;             /*Save the last button*/

    return false;                    /*No buffering now so no more data read*/
}
```

3.5 Tick interface

The LittlevGL needs a system tick to know the elapsed time for animation and other task.

You need to call the `lv_tick_inc(tick_period)` function periodically and tell the call period in milliseconds. For example, if called in every millisecond: `lv_tick_inc(1)`.

`lv_tick_inc` should be called in a higher priority routine than `lv_task_handler()` (e.g. in an interrupt) to precisely know the elapsed milliseconds even if the execution of `lv_task_handler` takes longer time.

With FreeRTOS `lv_tick_inc` can be called in `vApplicationTickHook`.

On Linux based operation system (e.g. on Raspberry) `lv_tick_inc` can be called in a thread:

```
void * tick_thread (void *args)
{
    while(1) {
        usleep(5*1000);    /*Sleep for 5 millisecond*/
        lv_tick_inc(5);    /*Tell LittlevGL that 5 milliseconds were elapsed*/
    }
}
```

3.6 Task Handler

To handle the tasks of LittlevGL you need to call `lv_task_handler()` periodically in one of the followings:

- `while(1)` of `main()` function
- timer interrupt periodically (low priority then `lv_tick_inc()`)
- an OS task periodically

The timing is not critical but it should be about 5 milliseconds to keep the system responsive.

Example:

```
while(1) {
    lv_task_handler();
    my_delay_ms(5);
}
```

3.7 Sleep management

The MCU can go to sleep when no user input happens. In this case the main `while(1)` should look like this:

```
while(1) {
    /*Normal operation (no sleep) in < 1 sec inactivity*/
    if(lv_disp_get_inactive_time(NULL) < 1000) {
        lv_task_handler();
    }
    /*Sleep after 1 sec inactivity*/
    else {
        timer_stop();    /*Stop the timer where lv_tick_inc() is called*/
        sleep();          /*Sleep the MCU*/
    }
    my_delay_ms(5);
}
```

You should also add these lines to your input device read function if a press happens:

```
lv_tick_inc(LV_REFR_PERIOD); /*Force task execution on wake-up*/
timer_start();               /*Restart the timer where lv_tick_inc() is called*/
lv_task_handler();           /*Call `lv_task_handler()` manually to process the
↪press event*/
```

In addition to `lv_disp_get_inactive_time()` you can check `lv_anim_count_running()` to see if every animations are finished.

3.8 Use with an operating system

LittlevGL is **not thread-safe** by default. Despite it, it's quite simple to use LittlevGL inside an operating system.

The **simple scenario** is to don't use the operating system's tasks but use `lv_tasks`. An *lv_task* is a function called periodically in `lv_task_handler`. In the *lv_task* you can get the state of the sensors, buffers, etc and call LittlevGL functions to refresh the GUI.

To create an *lv_task* use:

```
lv_task_create(my_func, period_ms, LV_TASK_PRIO_LOWEST/LOW/MID/HIGH/HIGHEST, custom_
↪ptr)
```

If you need to **use real tasks or threads** you need one mutex which should be taken before the call of `lv_task_handler` and released after it. In addition, you have to use to that mutex in other tasks and threads around every LittlevGL (`lv_...`) related function call and code. This way you can use LittlevGL in a real multitasking environment. Just use a mutex to avoid the concurrent calling of LittlevGL functions.

OVERVIEW

4.1 Objects

In the LittlevGL the **basic building blocks** of a user interface are the objects. For example a *Button*, *Label*, *Image*, *List*, *Chart* or *Text area*.

Check all the *Object types* here.

4.1.1 Object attributes

Basic attributes

The objects have basic attributes which are common independently from their type:

- Position
- Size
- Parent
- Drag enable
- Click enable etc.

You can set/get this attributes with `lv_obj_set_...` and `lv_obj_get_...` functions. For example:

```
/*Set basic object attributes*/  
lv_obj_set_size(btn1, 100, 50);      /*Button size*/  
lv_obj_set_pos(btn1, 20, 30);        /*Button position*/
```

To see all the available functions visit the Base object's *documentation*.

Specific attributes

The object types have special attributes too. For example, a slider has

- Min. max. values
- Current value
- Custom styles

For these attributes every object type have unique API functions. For example for a slider:

```
/*Set slider specific attributes*/
lv_slider_set_range(slider1, 0, 100);          /*Set min. and max. values*/
lv_slider_set_value(slider1, 40, LV_ANIM_ON);    /*Set the current value_
↪(position)*/
lv_slider_set_action(slider1, my_action);        /*Set a callback function*/
```

The API of the of the object types are described in their [Documentation](#) but you can also check the respective header files (e.g. `lv_objx/lv_slider.h`)

4.1.2 Object's working mechanisms

Parent-child structure

A parent object can be considered as the container of its children. Every object has exactly one parent object (except screens) but a parent can have unlimited number of children. There is no limitation for the type of the parent but there are typical parent (e.g. button) and typical child (e.g. label) objects.

Moving together

If the position of the parent is changed the children will move with the parent. Therefore all positions are relative to the parent.

The (0;0) coordinates mean the objects will remain in the top left-hand corner of the parent independently from the position of the parent.



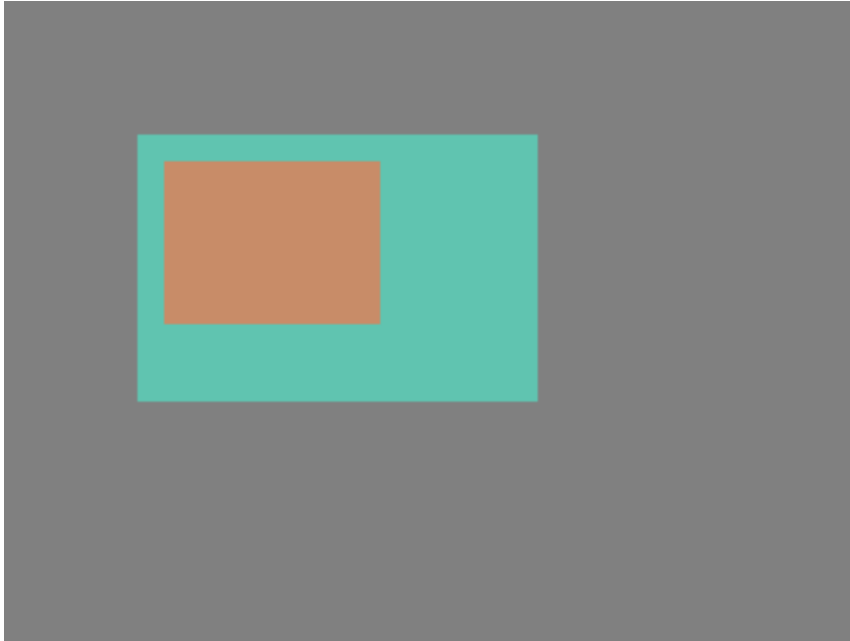
```
lv_obj_t * par = lv_obj_create(lv_scr_act(), NULL); /*Create a parent object on the_
↪current screen*/
lv_obj_set_size(par, 100, 80);                      /*Set the size of the_
↪parent*/
```

(continues on next page)

(continued from previous page)

```
lv_obj_t * obj1 = lv_obj_create(par, NULL);           /*Create an object on the_
↪previously created parent object*/
lv_obj_set_pos(obj1, 10, 10);                         /*Set the position of the new_
↪object*/
```

Modify the position of the parent:



```
lv_obj_set_pos(par, 50, 50);                         /*Move the parent. The child will move with it.*/
```

(For simplicity the adjusting of colors of the objects is not shown in the example.)

Visibility only on the parent

If a child partially or fully out of its parent then the parts outside will not be visible.



```
lv_obj_set_x(obj1, -30);           /*Move the child a little bit of the parent*/
```

Create - delete objects

In LittlevGL objects can be created and deleted dynamically in run-time. It means only the currently created objects consume RAM. For example, if you need a chart you can create it only when it is required and delete when its already not required.

Every objects type has its own **create** function with a unified prototype. It needs two parameters:

- a pointer the parent object. To create a screen give *NULL* as parent.
- optionally a pointer to an other object with the same type to copy it. Can be *NULL* to not copy an other object.

Independently from the object type a common variable type `lv_obj_t` is used. This pointer can be used later to set or get the attributes of the object.

The create functions look like this:

```
lv_obj_t * lv_<type>_create(lv_obj_t * parent, lv_obj_t * copy);
```

There is a common **delete** function for all object types. It deletes the object and all of its children.

```
void lv_obj_del(lv_obj_t * obj);
```

You can delete only the children of an object but leave the object itself “alive”:

```
void lv_obj_clean(lv_obj_t * obj);
```

Screen – the most basic parent

The screens are special objects which have no parent object. So it is created like:

```
lv_obj_t * scr1 = lv_obj_create(NULL, NULL);
```

Always there is an active screen on display. By default, the library creates and loads one. To get the currently active screen use the `lv_scr_act()` function to load new one use `lv_scr_load(scr1)`.

Screens can be created with any object type. For example, a *Base object* or an image to make a wallpaper.

Screens are created on the *default display*. The *default screen* is the lastly registered screen with `lv_disp_drv_register` (if there is only screen then that one) or you can explicitly selected display with `lv_disp_set_default(display)`. `lv_scr_act()` and `lv_scr_load()` operate on the currently default screen.

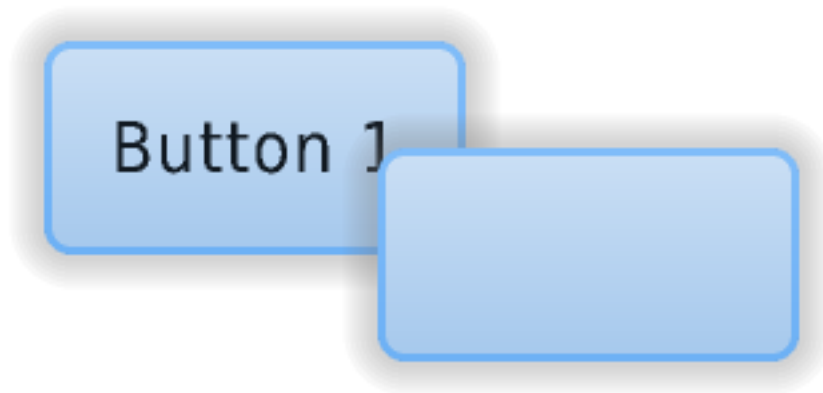
Visit [Multi display support](#) to learn more.

4.2 Layers

4.2.1 Order of creation

The earlier created object (and its children) will be drawn earlier (nearer to the background). In other words, the lastly created object will be on the top among its siblings. It is very important, the order is calculated among the objects on the same level (“siblings”).

Layers can be added easily by creating 2 objects (which can be transparent). Firstly ‘A’ and secondly ‘B’. ‘A’ and every object on it will be in the background and can be covered by ‘B’ and its children.



```
/*Create a screen*/
lv_obj_t * scr = lv_obj_create(NULL, NULL);
lv_scr_load(scr);          /*Load the screen*/

/*Create 2 buttons*/
lv_obj_t * btn1 = lv_btn_create(scr, NULL);          /*Create a button on the screen*/
lv_btn_set_fit(btn1, true, true);                    /*Enable to automatically set the
↪size according to the content*/
```

(continues on next page)

(continued from previous page)

```
lv_obj_set_pos(btn1, 60, 40);           /*Set the position of the_
↪button*/

lv_obj_t * btn2 = lv_btn_create(scr, btn1);    /*Copy the first button*/
lv_obj_set_pos(btn2, 180, 80);           /*Set the position of the button*/

/*Add labels to the buttons*/
lv_obj_t * label1 = lv_label_create(btn1, NULL);    /*Create a label on the first_
↪button*/
lv_label_set_text(label1, "Button 1");           /*Set the text of the label*/

lv_obj_t * label2 = lv_label_create(btn2, NULL);    /*Create a label on the_
↪second button*/
lv_label_set_text(label2, "Button 2");           /*Set the text of the_
↪label*/

/*Delete the second label*/
lv_obj_del(label2);
```

4.2.2 Bring to the foreground

There are several ways to bring an object to the foreground:

- Use `lv_obj_set_top(obj, true)`. If `obj` or any of its children is clicked then LittlevGL will automatically bring the object to the foreground. It works similarly to the windows on PC. When a window in the background is clicked it will come to the foreground automatically.
- Use `lv_obj_move_foreground(obj)` and `lv_obj_move_background(obj)` to explicitly tell the library to bring an object to the foreground or move to the background.
- When `lv_obj_set_parent(obj, new_parent)` is used `obj` will be on the foreground on the new parent.

4.2.3 Top and sys layer

There are two special layers called `layer_top` and `layer_sys`. Both of them is visible and the same on all screens of a display. `layer_top` is on top of “normal screen” and `layer_sys` is on top of `layer_top` too.

`layer_top` can be used by the user to create some content visible everywhere. For example a menu bar, a pop-up, etc. If the `click` attribute is enabled then `layer_top` will absorb all user click and acts as a modal.

```
lv_obj_set_click(lv_layer_top(), true);
```

`layer_sys` is used by LittlevGL. For example, it places the mouse cursor there to be sure it’s always visible.

4.3 Events

In LittlevGL events are triggered if something happens which might be interesting to the user. For example an object

- is clicked
- is dragged
- its value has changed, etc.

The user can assign a callback function to an object to see these event. In the practice it looks like this:

```
lv_obj_t * btn = lv_btn_create(lv_scr_act(), NULL);
lv_obj_set_event_cb(btn, my_event_cb);    /*Assign an event callback*/

...

static void my_event_cb(lv_obj_t * obj, lv_event_t event)
{
    switch(event) {
        case LV_EVENT_PRESSED:
            printf("Pressed\n");
            break;

        case LV_EVENT_SHORT_CLICKED:
            printf("Short clicked\n");
            break;

        case LV_EVENT_CLICKED:
            printf("Clicked\n");
            break;

        case LV_EVENT_LONG_PRESSED:
            printf("Long press\n");
            break;

        case LV_EVENT_LONG_PRESSED_REPEAT:
            printf("Long press repeat\n");
            break;

        case LV_EVENT_RELEASED:
            printf("Released\n");
            break;

        /*Etc.*/
    }
}
```

More objects can use the same *event callback*.

4.3.1 Event types

The following event types exist:

Generic events

Any object can receive these events independently from their type. I.e. these events are sent to Buttons, Labels, Sliders, etc.

Input device related

Sent when an object is pressed, released, etc by the user. They are used for *Keypad*, *Encoder* and *Button* input devices as well not only for *Pointers*. Visit the [Overview of input devices](#) section to learn more about them.

- **LV_EVENT_PRESSED** The object has been pressed

- **LV_EVENT_PRESSING** The object is being pressed (sent continuously while pressing)
- **LV_EVENT_PRESS_LOST** Still pressing but slid from the objects
- **LV_EVENT_SHORT_CLICKED** Released before `LV_INDEV_LONG_PRESS_TIME`. Not called if dragged.
- **LV_EVENT_LONG_PRESSED** Pressing for `LV_INDEV_LONG_PRESS_TIME` time. Not called if dragged.
- **LV_EVENT_LONG_PRESSED_REPEAT** Called after `LV_INDEV_LONG_PRESS_TIME` in every `LV_INDEV_LONG_PRESS_REPEAT_TIME` ms. Not called if dragged.
- **LV_EVENT_CLICKED** Called on release if not dragged (regardless to long press)
- **LV_EVENT_RELEASED** Called in every case when the object has been released even if it was dragged. Not called if slid from the object while pressing and released outside of the object. In this case, `LV_EVENT_PRESS_LOST` is sent.

Pointer related

These events are sent only by pointer-like input devices (E.g. mouse or touchpad)

- **LV_EVENT_DRAG_BEGIN** Dragging of the object has started
- **LV_EVENT_DRAG_END** Dragging finished (including drag throw)
- **LV_EVENT_DRAG_THROW_BEGIN** Drag throw started (released after drag with “momentum”)

Keypad and encoder related

These events are sent by keypad and encoder input devices. Learn more about *Groups* in [overview/indev](Input devices) section.

- **LV_EVENT_KEY** A *Key* is sent to the object. Typically when it was pressed or repeated after a long press
- **LV_EVENT_FOCUSED** The object is focused in its group
- **LV_EVENT_DEFOCUSED** The object is defocused in its group

General events

Other general events sent by the library.

- **LV_EVENT_DELETE** The object is being deleted. Free the related user-allocated data.

Special events

These events are specific to a partial object type.

- **LV_EVENT_VALUE_CHANGED** The object value has changed (e.g. for a *Slider*)
- **LV_EVENT_INSERT** Something is inserted to the object. (Typically to a *Text area*)
- **LV_EVENT_APPLY** “Ok”, “Apply” or similar specific button has clicked. (Typically from a *Keyboard* object)
- **LV_EVENT_CANCEL** “Close”, “Cancel” or similar specific button has clicked. (Typically from a *Keyboard* object)
- **LV_EVENT_REFRESH** Query to refresh the object. Never sent by the library but can be sent by the user.

To see exactly which events are used by an object type see the particular *Object type's documentation*.

4.3.2 Custom data

Some events might contain custom data. For example `LV_EVENT_VALUE_CHANGED` in some cases tells the new value. For more info see the particular *Object type's documentation*. To get the custom data in the event callback use `lv_event_get_data()`.

4.3.3 Send events manually

To manually send events to an object use `lv_event_send(obj, LV_EVENT_..., &custom_data)`.

It can be used for example to manually close a message box by simulating a button press:

```
/*Simulate the press of the first button (indexes start from zero)*/
uint32_t btn_id = 0;
lv_event_send(mbox, LV_EVENT_VALUE_CHANGED, &btn_id);
```

Or to ask refresh in a generic way.

```
lv_event_send(label, LV_EVENT_REFRESH, NULL);
```

4.4 Styles

Styles are used to set the appearance of the objects. A style is a structure variable with attributes like colors, paddings, opacity, etc.

There is common style type called `lv_style_t` for every object type.

Styles are assigned to the objects and by setting the fields of the `lv_style_t` variables you can influence the appearance of the objects using that style.

Important: The objects store only a pointer to a style so the style cannot be a local variable which is destroyed after the function exists. **You should use static, global or dynamically allocated variables.**

```
lv_style_t style_1;           /*OK! Global variables for styles are fine*/
static lv_style_t style_2;    /*OK! Static variables outside the functions are_
↪fine*/
void my_screen_create(void)
{
    static lv_style_t style_3; /*OK! Static variables in the functions are fine*/
    lv_style_t style_4;        /*WRONG! Styles can't be local variables*/

    ...
}
```

4.4.1 Use the styles

The objects have a *Main style* which determines the appearance of their background or main part. However, object types can have additional styles too.

Some object has only one style. E.g.

- Label
- Image
- Line, etc

For example, a slider has 3 styles:

- Background (main style)
- Indicator
- Know

Every object type has its own style set/get functions. For example

```
const lv_style_t * btn_style = lv_btn_get_style(btn, LV_BTN_STYLE_REL);
lv_btn_set_style(btn, LV_BTN_STYLE_REL, &new_style);
```

The styles supported by an object type (*LV_<OBJ_TYPE>STYLE<STYLE_TYPE>*) see the documentation of the particular *Object type*.

If you **modify a style which is already used** by one or more objects then the objects have to be notified about the style is changed. You have two options to do that:

```
/*Notify an object about its style is modified*/
void lv_obj_refresh_style(lv_obj_t * obj);

/*Notify all objects with a given style. (NULL to notify all objects)*/
void lv_obj_report_style_mod(void * style);
```

`lv_obj_report_style_mod` can refresh only the *Main styles*.

4.4.2 Inherit styles

If the *Main style* of an object is `NULL` then its style will be inherited from its parent's style. It makes easier to create a consistent design. Don't forget a style describes a lot of properties at the same time. So for example, if you set a button's style and create a label on it with `NULL` style then the label will be rendered according to the button's style. In other words, the button makes sure its children will look well on it.

Setting the `glass` style property will prevent inheriting that style. You should use it if the style is transparent so that its children use colors and others from its grandparent.

4.4.3 Style properties

A style has 5 main parts: common, body, text, image and line. An object will use those fields which are relevant to it. For example, *Lines* don't care about the *letter_space*. To see which fields are used by an object type see their *Documentation*.

The fields of a style structure are the followings:

Common properties

- **glass 1**: Do not inherit this style

Body style properties

Used by the rectangle-like objects

- **body.main_color** Main color (top color)
- **body.grad_color** Gradient color (bottom color)
- **body.radius** Corner radius. (set to `LV_RADIUS_CIRCLE` to draw circle)
- **body.opa** Opacity (0..255 or `LV_OPA_TRANSP`, `LV_OPA_10`, `LV_OPA_20` ... `LV_OPA_COVER`)
- **body.border.color** Border color
- **body.border.width** Border width
- **body.border.part** Border parts (`LV_BORDER_LEFT/RIGHT/TOP/BOTTOM/FULL` or 'OR'ed values)
- **body.border.opa** Border opacity (0..255 or `LV_OPA_TRANSP`, `LV_OPA_10`, `LV_OPA_20` ... `LV_OPA_COVER`)
- **body.shadow.color** Shadow color
- **body.shadow.width** Shadow width
- **body.shadow.type** Shadow type (`LV_SHADOW_BOTTOM/FULL`)
- **body.padding.top** Top padding
- **body.padding.bottom** Bottom padding
- **body.padding.left** Left padding
- **body.padding.right** Right padding
- **body.padding.inner** Inner padding (between content elements or children)

Text style properties

Used by the objects which show texts

- **text.color** Text color
- **text.sel_color** Selected text color
- **text.font** Pointer to a font
- **text.opa** Text opacity (0..255 or `LV_OPA_TRANSP`, `LV_OPA_10`, `LV_OPA_20` ... `LV_OPA_COVER*`)
- **text.letter_space** Letter space
- **text.line_space** Line space

Image style properties

Used by image-like objects or icons on objects

- **image.color** Color for image re-coloring based on the pixels brightness
- **image.intense** Re-color intensity (0..255 or `LV_OPA_TRANSP`, `LV_OPA_10`, `LV_OPA_20` ... `LV_OPA_COVER`)
- **image.opa** Image opacity (0..255 or `LV_OPA_TRANSP`, `LV_OPA_10`, `LV_OPA_20` ... `LV_OPA_COVER`)

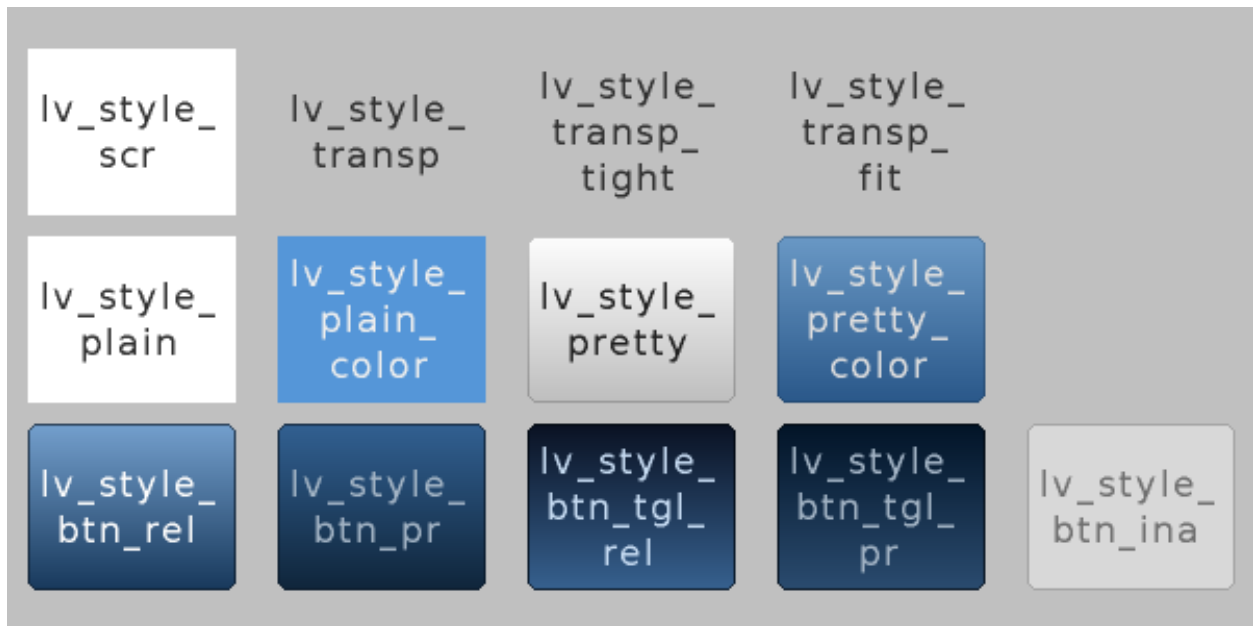
Line style properties

Used by objects containing lines or line-like elements

- **line.color** Line color
- **line.width** Line width
- **line.opa** Line opacity (0..255 or *LV_OPA_TRANSP*, *LV_OPA_10*, *LV_OPA_20* ... *LV_OPA_COVER*)

4.4.4 Built-in styles

There are several built-in styles in the library:



As you can see there is a style for screens, for buttons, plain and pretty styles and transparent styles as well.

The `lv_style_transp`, `lv_style_transp_fit` and `lv_style_transp_tight` differ only in paddings: for `lv_style_transp_tight` all padings are zero, for `lv_style_transp_fit` only hor and ver paddings are zero but has inner padding.

Important: Transparent built-in styles have *glass = 1* by default which means these styles (e.g. their colors) won't be inherited by children.

The built in styles are global `lv_style_t` variables. You can use them like:

```
lv_btn_set_style(obj, LV_BTN_STYLE_REL, &lv_style_btn_rel)
```

You can modify the built-in styles or you can create new styles. When creating new styles it is recommended to first copy a built-in style to be sure all fields are initialized with a proper value. The `lv_style_copy(&dest_style, &src_style)` can be used to copy styles.

4.4.5 Style animations

You change the styles with animations using `lv_style_anim_...()` function. Two styles are required to represent the *start* and *end* state, and a third style which will be animated. Here is an example to show how it works.

```
lv_anim_t a;
lv_style_anim_init(&a);                                     /*A basic_
↪initialization*/
lv_style_anim_set_styles(&a, &style_to_anim, &style_start, &style_end); /*Set the_
↪styles to use*/
lv_style_anim_set_time(&a, duration, delay);               /*Set the_
↪duration and delay*/
lv_style_anim_create(&a);                                   /*Create the_
↪animation*/
```

To see the whole API of style animations see `lv_core/lv_style.h`.

Here you can learn more about the [Animations](#).

4.4.6 Style example

The example below demonstrates the usage of styles.



Styles usage example in LittlevGL Embedded

Graphics Library

```
/*Create a style*/
static lv_style_t style1;
lv_style_copy(&style1, &lv_style_plain); /*Copy a built-in style to initialize the_
↪new style*/
style1.body.main_color = LV_COLOR_WHITE;
style1.body.grad_color = LV_COLOR_BLUE;
style1.body.radius = 10;
style1.body.border.color = LV_COLOR_GRAY;
style1.body.border.width = 2;
style1.body.border.opa = LV_OPA_50;
style1.body.padding.left = 5; /*Horizontal padding, used by the bar_
↪indicator below*/
style1.body.padding.right = 5;
style1.body.padding.top = 5; /*Vertical padding, used by the bar indicator_
↪below*/
style1.body.padding.bottom = 5;
style1.text.color = LV_COLOR_RED;

/*Create a simple object*/
lv_obj_t *obj1 = lv_obj_create(lv_scr_act(), NULL);
```

(continues on next page)

(continued from previous page)

```
lv_obj_set_style(obj1, &style1);           /*Apply the created style*/
lv_obj_set_pos(obj1, 20, 20);              /*Set the position*/

/*Create a label on the object. The label's style is NULL by default*/
lv_obj_t *label = lv_label_create(obj1, NULL);
lv_obj_align(label, NULL, LV_ALIGN_CENTER, 0, 0); /*Align the label to the
↪middle*/

/*Create a bar*/
lv_obj_t *bar1 = lv_bar_create(lv_scr_act(), NULL);
lv_bar_set_style(bar1, LV_BAR_STYLE_INDIC, &style1); /*Modify the indicator's
↪style*/
lv_bar_set_value(bar1, 70);                 /*Set the bar's value*/
```

4.4.7 Themes

To create styles for your GUI is challenging because you need a deeper understanding of the library and you need to have some design skills. In addition, it takes a lot of time to create so many styles.

To speed up the design part themes are introduced. A theme is a style collection which contains the required styles for every object type. For example 5 styles for buttons to describe their 5 possible states. Check the [Existing themes](#) or try some in the [Live demo](#) section.

To be more specific a theme is a structure variable which contains a lot of `lv_style_t *` fields. For buttons:

```
theme.btn.rel      /*Released button style*/
theme.btn.pr       /*Pressed button style*/
theme.btn.tgl_rel  /*Toggled released button style*/
theme.btn.tgl_pr   /*Toggled pressed button style*/
theme.btn.ina      /*Inactive button style*/
```

A theme can be initialized by: `lv_theme_<name>_init(hue, font)`. Where hue is a Hue value from [HSV color space](#) (0..360) and font is the font applied in the theme (NULL to use the LV_FONT_DEFAULT)

When a theme is initialized its styles can be used like this:



```
/*Create a default slider*/
lv_obj_t *slider = lv_slider_create(lv_scr_act(), NULL);
lv_slider_set_value(slider, 70);
lv_obj_set_pos(slider, 10, 10);

/*Initialize the alien theme with a redish hue*/
lv_theme_t *th = lv_theme_alien_init(10, NULL);

/*Create a new slider and apply the themes styles*/
slider = lv_slider_create(lv_scr_act(), NULL);
lv_slider_set_value(slider, 70);
```

(continues on next page)

(continued from previous page)

```
lv_obj_set_pos(slider, 10, 50);
lv_slider_set_style(slider, LV_SLIDER_STYLE_BG, th->slider.bg);
lv_slider_set_style(slider, LV_SLIDER_STYLE_INDIC, th->slider.indic);
lv_slider_set_style(slider, LV_SLIDER_STYLE_KNOB, th->slider.knob);
```

You can ask the library to automatically apply the styles from a theme when you create new objects. To do this use `lv_theme_set_current(th)`;

```
/*Initialize the alien theme with a redish hue*/
lv_theme_t *th = lv_theme_alien_init(10, NULL);
lv_theme_set_current(th);

/*Craete a slider. It will use the style from teh current theme.*/
slider = lv_slider_create(lv_scr_act(), NULL);
```

Themes can be enabled or disabled one by one in `lv_conf.h`.

Live update

By default if `lv_theme_set_current(th)` is called again it won't refresh the styles of the exisitin objects. To enable live update of themes enable `LV_THEME_LIVE_UPDATE` in `lv_conf.h`.

Live update will update only those objects whose style are from the theme, i.e. created after the first call of `lv_theme_set_current(th)` or the styles were set manually

4.5 Input devices

Assume porting is already read

4.5.1 Run time config

4.5.2 Pointers

cursor

API

4.5.3 Keypad and encoder

Groups

Keys

ENTER special

Keypads

Encoders

Edit and navigation mode

API

4.6 Displays

Assume porting is already read

4.6.1 Multi-display support

How why?

API

4.6.2 Run time config

4.7 Fonts

In LittlevGL fonts are collections of bitmaps and other informations required to render the images of the letters (glyph). A font is stored in a **lv_font_t** variable and can be set it in style's *text.font* field. For example:

```
my_style.text.font = &lv_font_roboto_28; /*Set a larger font*/
```

The fonts have a **bpp (Bit-Per-Pixel)** property. It shows how many bits are used to describe a pixel in the font. The value stored for a pixel determines the pixel's opacity. This way with higher *bpp* the edges of the letter can be smoother. The possible *bpp* values are 1, 2, 4 and 8 (higher value means better quality).

The *bpp* also affects the required memory size to store the font. E.g. *bpp* = 4 makes the font ~4 times greater compared to *bpp* = 1.

4.7.1 Unicode support

LittlevGL supports **UTF-8** encoded Unicode characters. You need to configure your editor to save your code/text as UTF-8 (usually this the default) and be sure `LV_TXT_ENC` is set to `LV_TXT_ENC_UTF8` in *lv_conf.h*. (This is the default value)

To test it try

```
lv_obj_t * labell = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(labell, LV_SYMBOL_OK);
```

If all works well a ✓ character should be displayed.

4.7.2 Built-in fonts



















































There are several built-in fonts in different sizes which can be enabled in *lv_conf.h* by *LV_FONT_...* defines:

- `LV_FONT_ROBOTO_12` 12 px
- `LV_FONT_ROBOTO_16` 16 px
- `LV_FONT_ROBOTO_22` 22 px
- `LV_FONT_ROBOTO_28` 28 px

The built-in fonts are **global variables** with names like `lv_font_roboto_16` for 16 px high font. To use them in a style just add a pointer to a font variable like shown above.

The built-in fonts have $bpp = 4$, contains the ASCII characters and uses the [Roboto](#) font.

In addition to the ASCII range, the following symbols are also added to the built-in fonts from the [FontAwesome](#) font.

	LV_SYMBOL_AUDIO
	LV_SYMBOL_VIDEO
	LV_SYMBOL_LIST
	LV_SYMBOL_OK
	LV_SYMBOL_CLOSE
	LV_SYMBOL_POWER
	LV_SYMBOL_SETTINGS
	LV_SYMBOL_TRASH
	LV_SYMBOL_HOME
	LV_SYMBOL_DOWNLOAD
	LV_SYMBOL_DRIVE
	LV_SYMBOL_REFRESH
	LV_SYMBOL_MUTE
	LV_SYMBOL_VOLUME_MID
	LV_SYMBOL_VOLUME_MAX
	LV_SYMBOL_IMAGE
	LV_SYMBOL_EDIT
	LV_SYMBOL_PREV
	LV_SYMBOL_PLAY
	LV_SYMBOL_PAUSE
	LV_SYMBOL_STOP
	LV_SYMBOL_NEXT
	LV_SYMBOL_EJECT
	LV_SYMBOL_LEFT
	LV_SYMBOL_RIGHT
	LV_SYMBOL_PLUS
	LV_SYMBOL_MINUS
	LV_SYMBOL_WARNING
	LV_SYMBOL_SHUFFLE
	LV_SYMBOL_UP
	LV_SYMBOL_DOWN
	LV_SYMBOL_LOOP
	LV_SYMBOL_DIRECTORY
	LV_SYMBOL_UPLOAD
	LV_SYMBOL_CALL
	LV_SYMBOL_CUT
	LV_SYMBOL_COPY
	LV_SYMBOL_SAVE
	LV_SYMBOL_CHARGE
	LV_SYMBOL_BELL
	LV_SYMBOL_KEYBOARD
	LV_SYMBOL_GPS
	LV_SYMBOL_FILE
	LV_SYMBOL_WIFI
	LV_SYMBOL_BATTERY_FULL
	LV_SYMBOL_BATTERY_3
	LV_SYMBOL_BATTERY_2
	LV_SYMBOL_BATTERY_1
	LV_SYMBOL_BATTERY_EMPTY
	LV_SYMBOL_BLUETOOTH

The symbols can be used as:

```
lv_label_set_text(my_label, LV_SYMBOL_OK);
```

Or with together with strings:

```
lv_label_set_text(my_label, LV_SYMBOL_OK "Apply");
```

Or more symbols together:

```
lv_label_set_text(my_label, LV_SYMBOL_OK LV_SYMBOL_WIFI LV_SYMBOL_PLAY);
```

4.7.3 Add new font

There are several ways to add a new font to your project:

1. The most simple way is to use the [Online font converter](#). Just set the parameters, click the *Convert* button, copy the font to your project and use it.
2. Use the [Offline font converter](#). (Requires Node.js to be installed)
3. If you want to create something like the built-in fonts (Roboto font and symbols) but in different size and/or ranges you can use the `built_in_font_gen.py` script in `lvgl/scripts/built_in_font` folder. (It requires Python and `lv_font_conv` to be installed)

To declare the font in a file use `LV_FONT_DECLARE(my_font_name)`.

To make to font globally available add them to `LV_FONT_CUSTOM_DECLARE` in `lv_conf.h`.

4.7.4 Add new symbols

The built-in symbols are created from [FontAwesome](#) font. To add new symbols from the [FontAwesome](#) font do the following steps:

1. Search symbol on <https://fontawesome.com>. For example the [USB symbol](#)
2. Open the [Online font converter](#) add [FontAwesome.ttf](#) and add the Unicode ID of the symbol to the range field. E.g. `0xf287` for the USB symbol. More symbols can be enumerated with `,`.
3. Convert the font and copy it to your project.
4. Convert the Unicode value to UTF8. You can do it e.g. on [this site](#). For `0xf287` the *Hex UTF-8 bytes* are `EF 8A 87`.
5. Create a define from the UTF8 values: `#define MY_USB_SYMBOL "\xEF\x8A\x87"`
6. Use the symbol as the built-in symbols. `lv_label_set_text(label, MY_USB_SYMBOL)`

4.7.5 Add a new font engine

LittlevGL's font interface is designed to be very flexible. You don't need to use LittlevGL's internal font engine but you can add your own. For example use [FreeType](#) to real-time render glyphs from TTF fonts or use an external flash to store the font's bitmap and read them when the library need them.

To do this a custom `lv_font_t` variable needs to be created:

```

/*Describe the properties of a font*/
lv_font_t my_font;
my_font.get_glyph_dsc = my_get_glyph_dsc_cb;           /*Set a callback to get info_
↳about glyphs*/
my_font.get_glyph_bitmap = my_get_glyph_bitmap_cb;     /*Set a callback to get bitmap of_
↳a glyph*/
my_font.line_height = height;                          /*The real line height where any_
↳text fits*/
my_font.base_line = base_line;                        /*Base line measured from the top_
↳of line_height*/
my_font.dsc = something_required;                      /*Store any implementation_
↳specific data here*/
my_font.user_data = user_data;                        /*Optionally some extra user_
↳data*/

...

/* Get info about glyph of `unicode_letter` in `font` font.
 * Store the result in `dsc_out`.
 * The next letter (`unicode_letter_next`) might be used to calculate the width_
↳required by this glyph (kerning)
 */
bool my_get_glyph_dsc_cb(const lv_font_t * font, lv_font_glyph_dsc_t * dsc_out,
↳uint32_t unicode_letter, uint32_t unicode_letter_next)
{
    /*Your code here*/

    /* Sotore the result.
     * For example ...
     */
    dsc_out->adv_w = 12;           /*Horizontal space required by the glyph in [px]*/
    dsc_out->box_h = 8;           /*Height of the bitmap in [px]*/
    dsc_out->box_w = 6;           /*Width of the bitmap in [px]*/
    dsc_out->ofs_x = 0;           /*X offset of the bitmap in [pf]*/
    dsc_out->ofs_y = 3;           /*Y ofset of the bitmap measured from the as line*/
    dsc_out->bpp = 2;             /*Bit per pixel: 1/2/4/8*/

    return true;                 /*true: glyph found; false: glyph was not found*/
}

/* Get the btmap of `unicode_letter` from `font`. */
const uint8_t * my_get_glyph_bitmap_cb(const lv_font_t * font, uint32_t unicode_
↳letter)
{
    /* Your code here */

    /* The bitmap should be a continuous bitstream where
     * each pixel is represented by `bpp` bits */

    return bitmap;              /*Or NULL if not found*/
}

```

4.8 Images

4.9 File system

4.10 Animations

You can automatically change the value of a variable between a start and an end value using animations. The animation will happen by the periodical call of an “animator” function with the corresponding value parameter.

The *animator* functions has the following prototype:

```
void func(void * var, lv_anim_var_t value);
```

This prototype is compatible with the majority of the *set* function of LittlevGL. For example `lv_obj_set_x(obj, value)` or `lv_obj_set_width(obj, value)`

4.10.1 Create an animation

To create an animation an `lv_anim_t` variable has to be initialized and configured with `lv_anim_set_...()` functions.

```
lv_anim_t a;
lv_anim_set_exec_cb(&a, btn1, lv_obj_set_x);    /*Set the animator function and
↪variable to animate*/
lv_anim_set_time(&a, duration, delay);
lv_anim_set_values(&a, start, end);             /*Set start and end values. E.g. 0,
↪150*/
lv_anim_set_path_cb(&a, lv_anim_path_linear);    /*Set path from `lv_anim_path_...`
↪functions or a custom one.*/
lv_anim_set_ready_cb(&a, ready_cb);             /*Set a callback to call then
↪animation is ready. (Optional)*/
lv_anim_set_playback(&a, wait_time);            /*Enable playback of teh animation
↪with `wait_time` delay*/
lv_anim_set_repeat(&a, wait_time);              /*Enable repeate of teh animation
↪with `wait_time` delay. Can be compined with playback*/
lv_anim_create(&a);                             /*Start the animation*/
```

You can apply **multiple different animations** on the same variable at the same time. For example animate the x and y coordinates with `lv_obj_set_x` and `lv_obj_set_y`. However, only one animation can exist with a given variable and function pair. Therefore `lv_anim_create()` will delete the already existing variable-function animations.

4.10.2 Animation path

You can determinate the **path of animation**. In the most simple case, it is linear which means the current value between *start* and *end* is changed linearly. A *path* is a function which calculates the next value to set based on the current state of the animation. Currently, there are the following built-in paths:

- `lv_anim_path_linear` linear animation
- `lv_anim_path_step` change in one step at the end
- `lv_anim_path_ease_in` slow at the beginning

- `lv_anim_path_ease_out` slow at the end
- `lv_anim_path_ease_in_out` slow at the beginning and end too
- `lv_anim_path_overshoot` overshoot the end value
- `lv_anim_path_bounce` bounce back a little from the end value (like hitting a wall)

4.10.3 Speed vs time

By default, you can set the animation time. But in some cases, the **animation speed** is more practical.

The `lv_anim_speed_to_time(speed, start, end)` function calculates the required time in milliseconds to reach the end value from a start value with the given speed. The speed is interpreted in *unit/sec* dimension. For example `lv_anim_speed_to_time(20, 0, 100)` will give 5000 milliseconds. For example in case of `lv_obj_set_x` *unit* is pixels so 20 means 20 *px/sec* speed.

4.10.4 Delete animations

You can **delete an animation** by `lv_anim_del(var, func)` by providing the animated variable and its animator function.

4.11 Drawing

With LittlevGL you don't need to draw anything manually. Just create objects (like buttons and labels), move and change them and LittlevGL will refresh and redraw what is required.

However, it might be useful to have a basic understanding of how drawing happens in LittlevGL.

The basic concept is to not draw directly to screen but draw to an internal buffer first and then copy that buffer to screen when the rendering is ready. It has two main advantages:

1. **Avoids flickering** while layers of the UI are drawn. E.g. when drawing a *background + button + text* each "stage" would be visible for a short time.
2. **It's faster** because when pixels are redrawn multiple times (e.g. background + button + text) it's faster to modify a buffer in RAM and finally write one pixel once than read/write a display directly on each pixel access. (e.g. via a display controller with SPI interface).

4.11.1 Buffering types

As you already might learn in the [Porting](#) section there are 3 types of buffering:

1. **One buffer** LittlevGL draws the content of the screen into a buffer and sends it to the display. The buffer can be smaller than the screen. In this case, the larger areas will be redrawn in multiple parts. If only small areas changes (e.g. button press) then only those areas will be refreshed.
2. **Two non-screen-sized buffers** having two buffers LittlevGL can draw into one buffer while the content of the other buffer is sent to display in the background. DMA or other hardware should be used to transfer the data to the display to let the CPU draw meanwhile. This way the rendering and refreshing of the display become parallel. Similarly to the *One buffer* LittlevGL will draw the display's content in chunks if the buffer is smaller than the area to refresh.

3. **Two screen-sized buffers.** In contrast to *Two non-screen-sized buffers* LittlevGL will always provide the whole screen's content not only chunks. This way the driver can simply change the address of the frame buffer to the buffer received from LittlevGL. Therefore this method works the best when the MCU has an LCD/TFT interface and the frame buffer is just a location in the RAM.

4.11.2 Mechanism of screen refreshing

1. Something happens on the GUI which requires redrawing. E.g. a button has been pressed, a chart has been changed or an animation happened, etc.
2. LittlevGL saves the changed object's old and new area into a buffer, called *Invalid area buffer*. For optimization in some cases objects are not added to the buffer:
 - Hidden objects are not added
 - Objects completely out of their parent are not added
 - Areas out of the parent are cropped to the parent's area
 - The object on other screens are not added
3. In every `LV_DISP_DEF_REFR_PERIOD` (set in *lv_conf.h*):
 - LittlevGL checks the invalid areas and joins the adjacent or intersecting areas
 - Takes the first joined area if it's smaller the *display buffer* then simply draws the areas content to the *display buffer*. If the area doesn't fit into the buffer draw as many lines as possible to the *display buffer*.
 - When the area is drawn call `flush_cb` from the display driver to refresh the display
 - If the area was larger than the buffer redraw the remaining parts too.
 - Do the same with all the joined areas.

While an area is redrawn the library searches the most top object which covers the area to redraw and starts to draw from that object. For example, if a button's label has changed the library will see that it's enough to draw the button under the text and it's not required to draw the background too.

The difference between buffer types regarding the drawing mechanism is the following:

1. **One buffer** LittlevGL needs to wait for `lv_disp_flush_ready()` (called at the end of `flush_cb`) before starting to redraw the next part.
2. **Two non-screen-sized buffers** LittlevGL can immediately draw to the second buffer when the first is sent to `flush_cb` because the flushing should be done by DMA (or similar hardware) in the background.
3. **Two screen-sized buffers** After calling `flush_cb` the first buffer if being displayed as frame buffer. Its content is copied to the second buffer and all the changes are drawn on top of it.

OBJECT TYPES

5.1 Base object (lv_obj)

5.2 Arc (lv_arc)

5.2.1 Overview

The *Arc* object **draws an arc** within **start and end angles** and with a given **thickness**.

To set the angles use the `lv_arc_set_angles(arc, start_angle, end_angle)` function. The zero degree is at the bottom of the object and the degrees are increasing in a counter-clockwise direction. The angles should be in `[0;360]` range.

The **width and height** of the *Arc* should be the **same**.

Currently the *Arc* object **does not support anti-aliasing**.

5.2.2 Styles

To set the style of an *Arc* object use `lv_arc_set_style(arc, LV_ARC_STYLE_MAIN, &style)`

- **line.rounded** make the endpoints rounded (opacity won't work properly if set to 1)
- **line.width** the thickness of the arc
- **line.color** the color of the arc.

5.2.3 Events

Only the [Generic events](#) are sent by the object type.

Learn more about [Events](#).

5.2.4 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.2.5 Example

C



Arc image

code

```
/*Create style for the Arcs*/
lv_style_t style;
lv_style_copy(&style, &lv_style_plain);
style.line.color = LV_COLOR_BLUE;           /*Arc color*/
style.line.width = 8;                       /*Arc width*/

/*Create an Arc*/
lv_obj_t * arc = lv_arc_create(lv_scr_act(), NULL);
lv_arc_set_style(arc, LV_ARC_STYLE_MAIN, &style);           /*Use the new style*/
lv_arc_set_angles(arc, 90, 60);
lv_obj_set_size(arc, 150, 150);
lv_obj_align(arc, NULL, LV_ALIGN_CENTER, 0, 0);

/*Copy the previous Arc and set different angles and size*/
arc = lv_arc_create(lv_scr_act(), arc);
lv_arc_set_angles(arc, 90, 20);
lv_obj_set_size(arc, 125, 125);
lv_obj_align(arc, NULL, LV_ALIGN_CENTER, 0, 0);

/*Copy the previous Arc and set different angles and size*/
arc = lv_arc_create(lv_scr_act(), arc);
lv_arc_set_angles(arc, 90, 310);
lv_obj_set_size(arc, 100, 100);
lv_obj_align(arc, NULL, LV_ALIGN_CENTER, 0, 0);
```

MicroPython

No examples yet.

5.2.6 API

Typedefs

```
typedef uint8_t lv_arc_style_t
```

Enums

```
enum [anonymous]
```

Values:

```
LV_ARC_STYLE_MAIN
```

Functions

```
lv_obj_t *lv_arc_create (lv_obj_t *par, const lv_obj_t *copy)
```

Create a arc objects Create a arc object

Return pointer to the created arc

Return pointer to the created arc

Parameters

- `par`: pointer to an object, it will be the parent of the new arc
- `copy`: pointer to a arc object, if not NULL then the new object will be copied from it

Parameters

- `par`: pointer to an object, it will be the parent of the new arc
- `copy`: pointer to a arc object, if not NULL then the new object will be copied from it

```
void lv_arc_set_angles (lv_obj_t *arc, uint16_t start, uint16_t end)
```

Set the start and end angles of an arc. 0 deg: bottom, 90 deg: right etc.

Parameters

- `arc`: pointer to an arc object
- `start`: the start angle [0..360]
- `end`: the end angle [0..360]

```
void lv_arc_set_style (lv_obj_t *arc, lv_arc_style_t type, const lv_style_t *style)
```

Set a style of a arc.

Parameters

- `arc`: pointer to arc object
- `type`: which style should be set
- `style`: pointer to a style

```
uint16_t lv_arc_get_angle_start (lv_obj_t *arc)
```

Get the start angle of an arc.

Return the start angle [0..360]

Parameters

- `arc`: pointer to an arc object

`uint16_t lv_arc_get_angle_end (lv_obj_t *arc)`

Get the end angle of an arc.

Return the end angle [0..360]

Parameters

- `arc`: pointer to an arc object

`const lv_style_t *lv_arc_get_style (const lv_obj_t *arc, lv_arc_style_t type)`

Get style of a arc.

Return style pointer to the style

Parameters

- `arc`: pointer to arc object
- `type`: which style should be get

`struct lv_arc_ext_t`

`#include <lv_arc.h>`

Public Members

`lv_coord_t angle_start`

`lv_coord_t angle_end`

5.3 Bar (lv_bar)

5.3.1 Overview

The Bar objects have got two main parts:

1. a **background** which is the object itself
2. an **indicator** which shape is similar to the background but its width/height can be adjusted.

The orientation of the bar can be **vertical or horizontal** according to the width/height ratio. Logically on horizontal bars the indicator width, on vertical bars the indicator height can be changed.

A **new value** can be set by: `lv_bar_set_value (bar, new_value, LV_ANIM_ON/OFF)`. The value is interpreted in **range** (minimum and maximum values) which can be modified with `lv_bar_set_range (bar, min, max)`. The default range is: 1..100.

The new value in `lv_bar_set_value` can be set with or without an **animation** depending on the last parameter (LV_ANIM_ON/OFF). The time of the animation can be adjusted by `lv_bar_set_anim_time (bar, 100)`. The time is in milliseconds unit.

The bar can be drawn **symmetrical** to zero (drawn from zero left or right) if it's enabled with `lv_bar_set_sym (bar, true)`

5.3.2 Styles

To set the style of an *Bar* object use `lv_bar_set_style (arc, LV_BAR_STYLE_MAIN, &style)`

- **LV_BAR_STYLE_BG** is an *Base object* therefore it uses its style elements. Its default style is: `lv_style_pretty`.

- **LV_BAR_STYLE_INDIC** is similar to the background. It uses the *left*, *right*, *top* and *bottom* paddings to keeps some space form the edges of the background. Its default style is: `lv_style_pretty_color`.

5.3.3 Events

Only the [Generic events](#) are sent by the object type.

Learn more about [Events](#).

5.3.4 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.3.5 Example

C



Bar image

code

```
/*Create a default bar*/
lv_obj_t * bar1 = lv_bar_create(lv_scr_act(), NULL);
lv_obj_set_size(bar1, 200, 30);
lv_obj_align(bar1, NULL, LV_ALIGN_IN_TOP_RIGHT, -20, 30);
lv_bar_set_value(bar1, 70);

/*Create a label right to the bar*/
lv_obj_t * bar1_label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(bar1_label, "Default");
lv_obj_align(bar1_label, bar1, LV_ALIGN_OUT_LEFT_MID, -10, 0);
```

(continues on next page)

(continued from previous page)

```

/*Create a bar and an indicator style*/
static lv_style_t style_bar;
static lv_style_t style_indic;

lv_style_copy(&style_bar, &lv_style_pretty);
style_bar.body.main_color = LV_COLOR_BLACK;
style_bar.body.grad_color = LV_COLOR_GRAY;
style_bar.body.radius = LV_RADIUS_CIRCLE;
style_bar.body.border.color = LV_COLOR_WHITE;

lv_style_copy(&style_indic, &lv_style_pretty);
style_indic.body.grad_color = LV_COLOR_GREEN;
style_indic.body.main_color = LV_COLOR_LIME;
style_indic.body.radius = LV_RADIUS_CIRCLE;
style_indic.body.shadow.width = 10;
style_indic.body.shadow.color = LV_COLOR_LIME;
style_indic.body.padding.hor = 3;          /*Make the indicator a little bit
↪smaller*/
style_indic.body.padding.ver = 3;

/*Create a second bar*/
lv_obj_t * bar2 = lv_bar_create(lv_scr_act(), bar1);
lv_bar_set_style(bar2, LV_BAR_STYLE_BG, &style_bar);
lv_bar_set_style(bar2, LV_BAR_STYLE_INDIC, &style_indic);
lv_obj_align(bar2, bar1, LV_ALIGN_OUT_BOTTOM_MID, 0, 30); /*Align below 'bar1'*/

/*Create a second label*/
lv_obj_t * bar2_label = lv_label_create(lv_scr_act(), bar1_label);
lv_label_set_text(bar2_label, "Modified");
lv_obj_align(bar2_label, bar2, LV_ALIGN_OUT_LEFT_MID, -10, 0);

```

MicroPython

No examples yet.

5.3.6 API

Defines

LV_BAR_ANIM_STATE_START

LV_BAR_ANIM_STATE_END

LV_BAR_ANIM_STATE_INV

LV_BAR_ANIM_STATE_NORM

Typedefs

typedef uint8_t lv_bar_style_t

Enums

enum [anonymous]

Values:

LV_BAR_STYLE_BG

LV_BAR_STYLE_INDIC

Functions

lv_obj_t *lv_bar_create (lv_obj_t *par, const lv_obj_t *copy)

Create a bar objects

Return pointer to the created bar

Parameters

- par: pointer to an object, it will be the parent of the new bar
- copy: pointer to a bar object, if not NULL then the new object will be copied from it

void lv_bar_set_value (lv_obj_t *bar, int16_t value, lv_anim_enable_t anim)

Set a new value on the bar Set a new value on the bar

Parameters

- bar: pointer to a bar object
- value: new value
- anim: LV_ANIM_ON: set the value with an animation; LV_ANIM_OFF: change the value immediately

Parameters

- bar: pointer to a bar object
- value: new value
- anim: LV_ANIM_ON: set the value with an animation; LV_ANIM_OFF: change the value immediately

void lv_bar_set_range (lv_obj_t *bar, int16_t min, int16_t max)

Set minimum and the maximum values of a bar

Parameters

- bar: pointer to the bar object
- min: minimum value
- max: maximum value

void lv_bar_set_sym (lv_obj_t *bar, bool en)

Make the bar symmetric to zero. The indicator will grow from zero instead of the minimum position.

Parameters

- bar: pointer to a bar object
- en: true: enable disable symmetric behavior; false: disable

void lv_bar_set_anim_time (lv_obj_t *bar, uint16_t anim_time)

Set the animation time of the bar

Parameters

- `bar`: pointer to a bar object
- `anim_time`: the animation time in milliseconds.

void **lv_bar_set_style** (lv_obj_t **bar*, *lv_bar_style_t* type, const lv_style_t **style*)

Set a style of a bar

Parameters

- `bar`: pointer to a bar object
- `type`: which style should be set
- `style`: pointer to a style

int16_t **lv_bar_get_value** (const lv_obj_t **bar*)

Get the value of a bar

Return the value of the bar

Parameters

- `bar`: pointer to a bar object

int16_t **lv_bar_get_min_value** (const lv_obj_t **bar*)

Get the minimum value of a bar

Return the minimum value of the bar

Parameters

- `bar`: pointer to a bar object

int16_t **lv_bar_get_max_value** (const lv_obj_t **bar*)

Get the maximum value of a bar

Return the maximum value of the bar

Parameters

- `bar`: pointer to a bar object

bool **lv_bar_get_sym** (lv_obj_t **bar*)

Get whether the bar is symmetric or not.

Return true: symmetric is enabled; false: disable

Parameters

- `bar`: pointer to a bar object

uint16_t **lv_bar_get_anim_time** (lv_obj_t **bar*, uint16_t *anim_time*)

Get the animation time of the bar

Return the animation time in milliseconds.

Parameters

- `bar`: pointer to a bar object

const lv_style_t ***lv_bar_get_style** (const lv_obj_t **bar*, *lv_bar_style_t* type)

Get a style of a bar

Return style pointer to a style

Parameters

- `bar`: pointer to a bar object
- `type`: which style should be get

```
struct lv_bar_ext_t
#include <lv_bar.h>
```

Public Members

```
int16_t cur_value
int16_t min_value
int16_t max_value
lv_anim_value_t anim_start
lv_anim_value_t anim_end
lv_anim_value_t anim_state
lv_anim_value_t anim_time
uint8_t sym
const lv_style_t *style_indic
```

5.4 Button (lv_btn)

5.4.1 Overview

Buttons are simple rectagnle-like objects but they change their style and state when they are pressed or released.

Buttons can be in one of the **five possible states**:

- **LV_BTN_STATE_REL** Released state
- **LV_BTN_STATE_PR** Pressed state
- **LV_BTN_STATE_TGL_REL** Toggled released state
- **LV_BTN_STATE_TGL_PR** Toggled pressed state
- **LV_BTN_STATE_INA** Inactive state

The buttons can be configured as **toggle button** with `lv_btn_set_toggle(btn, true)`. In this case on release, the button goes to toggled released state.

You can set the button's state manually with `lv_btn_set_state(btn, LV_BTN_STATE_TGL_REL)`.

Similarly to [Containers](#) buttons also have **layout** and **fit** attributes.

- `lv_btn_set_layout(btn, LV_LAYOUT_...)` set a layout. The default is `LV_LAYOUT_CENTER`. So if you add a label then it will be automatically aligned to the middle and can't be moved with `lv_obj_set_pos()`. You can disable the layout with `lv_btn_set_layout(btn, LV_LAYOUT_OFF)`
- `lv_btn_set_fit/fit2/fit4(btn, LV_FIT_...)` enables to set the button width and/or height automatically according to the children, parent and fit type.

5.4.2 Styles

A button can have 5 independent styles for the 5 state. You can set them via: `lv_btn_set_style(btn, LV_BTN_STYLE_..., &style)`. The styles use the `style.body` properties.

- **LV_BTN_STYLE_REL** style of the released state. Default: `lv_style_btn_rel`
- **LV_BTN_STYLE_PR** style of the pressed state. Default: `lv_style_btn_pr`
- **LV_BTN_STYLE_TGL_REL** style of the toggled released state. Default: `lv_style_btn_tgl_rel`
- **LV_BTN_STYLE_TGL_PR** style of the toggled pressed state. Default: `lv_style_btn_tgl_pr`
- **LV_BTN_STYLE_INA** style of the inactive state. Default: `lv_style_btn_ina`

When labels are created on a button, it's a good practice to set the button's `style.text` properties too. Because labels have `style = NULL` by default they inherit the parent's (button) style. Hence you don't need to create a new style for the label.

5.4.3 Events

Besides the [Generic events](#) the following [Special events](#) are sent by the buttons:

- **LV_EVENT_VALUE_CHANGED** sent when the button is toggled.

Note that the generic input device related events (like `LV_EVENT_PRESSED`) are sent in inactive state too. You need to check the state with `lv_btn_get_state(btn)` to ignore the events from inactive buttons.

Learn more about [Events](#).

5.4.4 Keys

The following *Keys* are processed by the Buttons:

- **LV_KEY_RIGHT/UP** Go to toggled state if toggling is enabled
- **LV_KEY_LEFT/DOWN** Go to non-toggled state if toggling is enabled

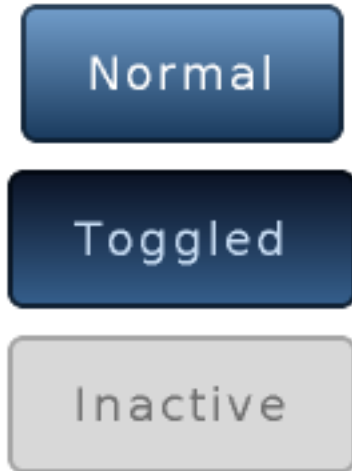
Note that, as usual, the state of `LV_KEY_ENTER` is translated to `LV_EVENT_PRESSED/PRESSING/RELEASED` etc.

Learn more about [Keys](#).

5.4.5 Example

C

Default buttons



Button image

code

```
static lv_res_t btn_click_action(lv_obj_t * btn)
{
    uint8_t id = lv_obj_get_free_num(btn);

    printf("Button %d is released\n", id);

    /* The button is released.
     * Make something here */

    return LV_RES_OK; /*Return OK if the button is not deleted*/
}

.
.
.

/*Create a title label*/
lv_obj_t * label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label, "Default buttons");
lv_obj_align(label, NULL, LV_ALIGN_IN_TOP_MID, 0, 5);

/*Create a normal button*/
lv_obj_t * btn1 = lv_btn_create(lv_scr_act(), NULL);
lv_cont_set_fit(btn1, true, true); /*Enable resizing horizontally and vertically*/
lv_obj_align(btn1, label, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
lv_obj_set_free_num(btn1, 1); /*Set a unique number for the button*/
lv_btn_set_action(btn1, LV_BTN_ACTION_CLICK, btn_click_action);

/*Add a label to the button*/
label = lv_label_create(btn1, NULL);
lv_label_set_text(label, "Normal");
```

(continues on next page)

(continued from previous page)

```

/*Copy the button and set toggled state. (The release action is copied too)*/
lv_obj_t * btn2 = lv_btn_create(lv_scr_act(), btn1);
lv_obj_align(btn2, btn1, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
lv_btn_set_state(btn2, LV_BTN_STATE_TGL_REL); /*Set toggled state*/
lv_obj_set_free_num(btn2, 2); /*Set a unique number for the button*/

/*Add a label to the toggled button*/
label = lv_label_create(btn2, NULL);
lv_label_set_text(label, "Toggled");

/*Copy the button and set inactive state.*/
lv_obj_t * btn3 = lv_btn_create(lv_scr_act(), btn1);
lv_obj_align(btn3, btn2, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
lv_btn_set_state(btn3, LV_BTN_STATE_INA); /*Set inactive state*/
lv_obj_set_free_num(btn3, 3); /*Set a unique number for the button*/

/*Add a label to the inactive button*/
label = lv_label_create(btn3, NULL);
lv_label_set_text(label, "Inactive");

```

MicroPython

No examples yet.

5.4.6 API

Typedefs

```

typedef uint8_t lv_btn_state_t
typedef uint8_t lv_btn_style_t

```

Enums

enum [anonymous]

Values:

```

LV_BTN_STATE_REL
LV_BTN_STATE_PR
LV_BTN_STATE_TGL_REL
LV_BTN_STATE_TGL_PR
LV_BTN_STATE_INA
LV_BTN_STATE_NUM

```

enum [anonymous]

Values:

```

LV_BTN_STYLE_REL
LV_BTN_STYLE_PR

```

LV_BTN_STYLE_TGL_REL
LV_BTN_STYLE_TGL_PR
LV_BTN_STYLE_INA

Functions

lv_obj_t *lv_btn_create (lv_obj_t *par, const lv_obj_t *copy)

Create a button objects

Return pointer to the created button

Parameters

- par: pointer to an object, it will be the parent of the new button
- copy: pointer to a button object, if not NULL then the new object will be copied from it

void lv_btn_set_toggle (lv_obj_t *btn, bool tgl)

Enable the toggled states. On release the button will change from/to toggled state. Enable the toggled states

Parameters

- btn: pointer to a button object
- tgl: true: enable toggled states, false: disable

Parameters

- btn: pointer to a button object
- tgl: true: enable toggled states, false: disable

void lv_btn_set_state (lv_obj_t *btn, lv_btn_state_t state)

Set the state of the button

Parameters

- btn: pointer to a button object
- state: the new state of the button (from lv_btn_state_t enum)

void lv_btn_toggle (lv_obj_t *btn)

Toggle the state of the button (ON->OFF, OFF->ON)

Parameters

- btn: pointer to a button object

static void lv_btn_set_layout (lv_obj_t *btn, lv_layout_t layout)

Set the layout on a button

Parameters

- btn: pointer to a button object
- layout: a layout from 'lv_cont_layout_t'

static void lv_btn_set_fit4 (lv_obj_t *btn, lv_fit_t left, lv_fit_t right, lv_fit_t top, lv_fit_t bottom)

Set the fit policy in all 4 directions separately. It tell how to change the button size automatically.

Parameters

- btn: pointer to a button object
- left: left fit policy from lv_fit_t

- `right`: right fit policy from `lv_fit_t`
- `top`: bottom fit policy from `lv_fit_t`
- `bottom`: bottom fit policy from `lv_fit_t`

static void `lv_btn_set_fit2` (`lv_obj_t` **btn*, `lv_fit_t` *hor*, `lv_fit_t` *ver*)

Set the fit policy horizontally and vertically separately. It tell how to change the button size automatically.

Parameters

- `btn`: pointer to a button object
- `hor`: horizontal fit policy from `lv_fit_t`
- `ver`: vertical fit policy from `lv_fit_t`

static void `lv_btn_set_fit` (`lv_obj_t` **cont*, `lv_fit_t` *fit*)

Set the fit policy in all 4 direction at once. It tell how to change the button size automatically.

Parameters

- `btn`: pointer to a button object
- `fit`: fit policy from `lv_fit_t`

void `lv_btn_set_ink_in_time` (`lv_obj_t` **btn*, `uint16_t` *time*)

Set time of the ink effect (draw a circle on click to animate in the new state)

Parameters

- `btn`: pointer to a button object
- `time`: the time of the ink animation

void `lv_btn_set_ink_wait_time` (`lv_obj_t` **btn*, `uint16_t` *time*)

Set the wait time before the ink disappears

Parameters

- `btn`: pointer to a button object
- `time`: the time of the ink animation

void `lv_btn_set_ink_out_time` (`lv_obj_t` **btn*, `uint16_t` *time*)

Set time of the ink out effect (animate to the released state)

Parameters

- `btn`: pointer to a button object
- `time`: the time of the ink animation

void `lv_btn_set_style` (`lv_obj_t` **btn*, `lv_btn_style_t` *type*, `const` `lv_style_t` **style*)

Set a style of a button. Set a style of a button

Parameters

- `btn`: pointer to button object
- `type`: which style should be set
- `style`: pointer to a style

Parameters

- `btn`: pointer to a button object
- `type`: which style should be set

- `style`: pointer to a style

`lv_btn_state_t lv_btn_get_state (const lv_obj_t *btn)`

Get the current state of the button

Return the state of the button (from `lv_btn_state_t` enum)

Parameters

- `btn`: pointer to a button object

`bool lv_btn_get_toggle (const lv_obj_t *btn)`

Get the toggle enable attribute of the button

Return true: toggle enabled, false: disabled

Parameters

- `btn`: pointer to a button object

`static lv_layout_t lv_btn_get_layout (const lv_obj_t *btn)`

Get the layout of a button

Return the layout from 'lv_cont_layout_t'

Parameters

- `btn`: pointer to button object

`static lv_fit_t lv_btn_get_fit_left (const lv_obj_t *btn)`

Get the left fit mode

Return an element of `lv_fit_t`

Parameters

- `btn`: pointer to a button object

`static lv_fit_t lv_btn_get_fit_right (const lv_obj_t *btn)`

Get the right fit mode

Return an element of `lv_fit_t`

Parameters

- `btn`: pointer to a button object

`static lv_fit_t lv_btn_get_fit_top (const lv_obj_t *btn)`

Get the top fit mode

Return an element of `lv_fit_t`

Parameters

- `btn`: pointer to a button object

`static lv_fit_t lv_btn_get_fit_bottom (const lv_obj_t *btn)`

Get the bottom fit mode

Return an element of `lv_fit_t`

Parameters

- `btn`: pointer to a button object

`uint16_t lv_btn_get_ink_in_time (const lv_obj_t *btn)`

Get time of the ink in effect (draw a circle on click to animate in the new state)

Return the time of the ink animation

Parameters

- `btn`: pointer to a button object

`uint16_t lv_btn_get_ink_wait_time(const lv_obj_t *btn)`

Get the wait time before the ink disappears

Return the time of the ink animation

Parameters

- `btn`: pointer to a button object

`uint16_t lv_btn_get_ink_out_time(const lv_obj_t *btn)`

Get time of the ink out effect (animate to the releases state)

Return the time of the ink animation

Parameters

- `btn`: pointer to a button object

`const lv_style_t *lv_btn_get_style(const lv_obj_t *btn, lv_btn_style_t type)`

Get style of a button. Get a style of a button

Return style pointer to the style

Return style pointer to a style

Parameters

- `btn`: pointer to button object
- `type`: which style should be get

Parameters

- `btn`: pointer to a button object
- `type`: which style should be get

`struct lv_btn_ext_t`
`#include <lv_btn.h>`

Public Members

`lv_cont_ext_t cont`

`const lv_style_t *styles[LV_BTN_STATE_NUM]`

`uint16_t ink_in_time`

`uint16_t ink_wait_time`

`uint16_t ink_out_time`

`lv_btn_state_t state`

`uint8_t toggle`

5.5 Button matrix (lv_btm)

5.5.1 Overview

The Button Matrix objects can display **multiple buttons** in rows and columns.

The Button matrix object is very light weighted because the buttons are not really created just drawn on the fly. This way 1 button uses only 8 extra byte instead of the ~100-150 byte size of a normal [Button](#) object.

The buttons have texts on them which can be specified a descriptor string array, called *map*. The map can be set with `lv_btm_set_map(btm, my_map)`.

The **declaration of a map** looks like `const char * map[] = {"btn1", "btn2", "btn3", ""}`. Note that **the last element has to be an empty string!**

Use `"\n"` in the map to make **line break**. E.g. `{"btn1", "btn2", "\n", "btn3", ""}`. The button's width is recalculated in every line.

The buttons width can be set relative to the other button in the same line with `lv_btm_set_btn_width(btm, btn_id, width)` E.g. in a line with two buttons: *btn 1 width = 1* and *btn 2 width = 2*, *btn 1* will have 33 % width and *btn 2* will have 66 % width.

In addition to width each button can be customized with following parameters:

- **LV_BTMM_CTRL_HIDDEN** make a button hidden
- **LV_BTMM_CTRL_NO_REPEAT** disable repeating when the button is long pressed
- **LV_BTMM_CTRL_INACTIVE** make a button inactive
- **LV_BTMM_CTRL_TGL_ENABLE** enable toggling of a button
- **LV_BTMM_CTRL_TGL_STATE** set the toggle state
- **LV_BTMM_CTRL_CLICK_TRIG** if 0 the button will react on press, if 1 will react on release

The set or clear a button's control attribute use `lv_btm_set_btn_ctrl(btm, btn_id, LV_BTMM_CTRL_...)` and `lv_btm_clear_btn_ctrl(btm, btn_id, LV_BTMM_CTRL_...)` respectively. More `LV_BTMM_CTRL_...` values can be *Ored*

The set/clear the same control attribute for all buttons of a button matrix use `lv_btm_set_btn_ctrl_all(btm, btn_id, LV_BTMM_CTRL_...)` and `lv_btm_clear_btn_ctrl_all(btm, btn_id, LV_BTMM_CTRL_...)`.

The *“One toggle”* feature can be enable with `lv_btm_set_one_toggle(btm, true)` to allow only one toggled button at once.

The set a control map for a button matrix (similarly to the map for the text) use `lv_btm_set_ctrl_map(btm, ctrl_map)`. An element of `ctrl_map` should look like `ctrl_map[0] = width | LV_BTMM_CTRL_NO_REPEAT | LV_BTMM_CTRL_TGL_ENABLE`. The number of elements should be equal to the number of buttons (excluding new lines).

The **texts** on the button can be **recolor** similarly to the recolor feature for [Label](#) object. To enable it use `lv_btm_set_recolor(btm, true)`. After that a button with `#FF0000` Red# text will be red.

5.5.2 Styles

The Button matrix works with 6 styles: a background and 5 button styles for each states. You can set the styles with `lv_btm_set_style(btn, LV_BTMM_STYLE_..., &style)`. The background and the buttons use the `style.body` properties. The labels use the `style.text` properties of the button styles.

- **LV_BTNM_STYLE_BG** Background style. Uses all *style.body* properties including *padding*. Default: *lv_style_pretty*
- **LV_BTNM_STYLE_BTN_REL** style of the released buttons. Default: *lv_style_btn_rel*
- **LV_BTNM_STYLE_BTN_PR** style of the pressed buttons. Default: *lv_style_btn_pr*
- **LV_BTNM_STYLE_BTN_TGL_REL** style of the toggled released buttons. Default: *lv_style_btn_tgl_rel*
- **LV_BTNM_STYLE_BTN_TGL_PR** style of the toggled pressed buttons. Default: *lv_style_btn_tgl_pr*
- **LV_BTNM_STYLE_BTN_INA** style of the inactive buttons. Default: *lv_style_btn_ina*

5.5.3 Events

Besides the [Generic events](#) the following [Special events](#) are sent by the button matrices:

- **LV_EVENT_VALUE_CHANGED** sent when the button is pressed/released or repeated after long press. The event data is set to ID of the pressed/released button.

Learn more about [Events](#).

##Keys

The following *Keys* are processed by the Buttons:

- **LV_KEY_RIGHT/UP/LEFT/RIGHT** To navigate among the buttons and elect one
- **LV_KEY_ENTER** To press/release the selected button

Learn more about [Keys](#).

5.5.4 Example

C



code


```

/*Called when a button is released ot long pressed*/
static lv_res_t btnm_action(lv_obj_t * btnm, const char *txt)
{
    printf("Button: %s released\n", txt);

    return LV_RES_OK; /*Return OK because the button matrix is not deleted*/
}

.
.
.

/*Create a button descriptor string array*/
static const char * btnm_map[] = {"1", "2", "3", "4", "5", "\n",
                                   "6", "7", "8", "9", "0", "\n",
                                   "\202Action1", "Action2", ""};

/*Create a default button matrix*/
lv_obj_t * btnm1 = lv_btnm_create(lv_scr_act(), NULL);
lv_btnm_set_map(btnm1, btnm_map);
lv_btnm_set_action(btnm1, btnm_action);
lv_obj_set_size(btnm1, LV_HOR_RES, LV_VER_RES / 2);

/*Create a new style for the button matrix back ground*/
static lv_style_t style_bg;
lv_style_copy(&style_bg, &lv_style_plain);
style_bg.body.main_color = LV_COLOR_SILVER;
style_bg.body.grad_color = LV_COLOR_SILVER;
style_bg.body.padding.hor = 0;
style_bg.body.padding.ver = 0;
style_bg.body.padding.inner = 0;

/*Create 2 button styles*/
static lv_style_t style_btn_rel;
static lv_style_t style_btn_pr;
lv_style_copy(&style_btn_rel, &lv_style_btn_rel);
style_btn_rel.body.main_color = LV_COLOR_MAKE(0x30, 0x30, 0x30);
style_btn_rel.body.grad_color = LV_COLOR_BLACK;
style_btn_rel.body.border.color = LV_COLOR_SILVER;
style_btn_rel.body.border.width = 1;
style_btn_rel.body.border.opa = LV_OPA_50;
style_btn_rel.body.radius = 0;

lv_style_copy(&style_btn_pr, &style_btn_rel);
style_btn_pr.body.main_color = LV_COLOR_MAKE(0x55, 0x96, 0xd8);
style_btn_pr.body.grad_color = LV_COLOR_MAKE(0x37, 0x62, 0x90);
style_btn_pr.text.color = LV_COLOR_MAKE(0xbb, 0xd5, 0xf1);

/*Create a second button matrix with the new styles*/
lv_obj_t * btnm2 = lv_btnm_create(lv_scr_act(), btnm1);
lv_btnm_set_style(btnm2, LV_BTNM_STYLE_BG, &style_bg);
lv_btnm_set_style(btnm2, LV_BTNM_STYLE_BTN_REL, &style_btn_rel);

```

(continues on next page)

(continued from previous page)

```
lv_btmn_set_style(btmn2, LV_BTNM_STYLE_BTN_PR, &style_btn_pr);
lv_obj_align(btmn2, btmn1, LV_ALIGN_OUT_BOTTOM_MID, 0, 0);
```

MicroPython

No examples yet.

5.5.5 API

Defines

LV_BTNM_WIDTH_MASK

LV_BTNM_BTN_NONE

Typedefs

```
typedef uint16_t lv_btmn_ctrl_t
```

```
typedef uint8_t lv_btmn_style_t
```

Enums

enum [anonymous]

Values:

LV_BTNM_CTRL_HIDDEN = 0x0008

LV_BTNM_CTRL_NO_REPEAT = 0x0010

LV_BTNM_CTRL_INACTIVE = 0x0020

LV_BTNM_CTRL_TGL_ENABLE = 0x0040

LV_BTNM_CTRL_TGL_STATE = 0x0080

LV_BTNM_CTRL_CLICK_TRIG = 0x0100

enum [anonymous]

Values:

LV_BTNM_STYLE_BG

LV_BTNM_STYLE_BTN_REL

LV_BTNM_STYLE_BTN_PR

LV_BTNM_STYLE_BTN_TGL_REL

LV_BTNM_STYLE_BTN_TGL_PR

LV_BTNM_STYLE_BTN_INA

Functions

`lv_obj_t *lv_btm_create (lv_obj_t *par, const lv_obj_t *copy)`

Create a button matrix objects

Return pointer to the created button matrix

Parameters

- `par`: pointer to an object, it will be the parent of the new button matrix
- `copy`: pointer to a button matrix object, if not NULL then the new object will be copied from it

void `lv_btm_set_map (const lv_obj_t *btm, const char *map[])`

Set a new map. Buttons will be created/deleted according to the map. The button matrix keeps a reference to the map and so the string array must not be deallocated during the life of the matrix.

Parameters

- `btm`: pointer to a button matrix object
- `map`: pointer a string array. The last string has to be: "". Use "\n" to make a line break.

void `lv_btm_set_ctrl_map (const lv_obj_t *btm, const lv_btm_ctrl_t ctrl_map[])`

Set the button control map (hidden, disabled etc.) for a button matrix. The control map array will be copied and so may be deallocated after this function returns.

Parameters

- `btm`: pointer to a button matrix object
- `ctrl_map`: pointer to an array of `lv_btm_ctrl_t` control bytes. The length of the array and position of the elements must match the number and order of the individual buttons (i.e. excludes newline entries). An element of the map should look like e.g.: `ctrl_map[0] = width | LV_BTNM_CTRL_NO_REPEAT | LV_BTNM_CTRL_TGL_ENABLE`

void `lv_btm_set_pressed (const lv_obj_t *btm, uint16_t id)`

Set the pressed button i.e. visually highlight it. Mainly used a when the btm is in a group to show the selected button

Parameters

- `btm`: pointer to button matrix object
- `id`: index of the currently pressed button (`LV_BTNM_BTN_NONE` to unpress)

void `lv_btm_set_style (lv_obj_t *btm, lv_btm_style_t type, const lv_style_t *style)`

Set a style of a button matrix

Parameters

- `btm`: pointer to a button matrix object
- `type`: which style should be set
- `style`: pointer to a style

void `lv_btm_set_recolor (const lv_obj_t *btm, bool en)`

Enable recoloring of button's texts

Parameters

- `btm`: pointer to button matrix object
- `en`: true: enable recoloring; false: disable

void **lv_btnm_set_btn_ctrl** (**const** lv_obj_t **btnm*, uint16_t *btn_id*, lv_btnm_ctrl_t *ctrl*)
 Set the attributes of a button of the button matrix

Parameters

- *btnm*: pointer to button matrix object
- *btn_id*: 0 based index of the button to modify. (Not counting new lines)

void **lv_btnm_clear_btn_ctrl** (**const** lv_obj_t **btnm*, uint16_t *btn_id*, lv_btnm_ctrl_t *ctrl*)
 Clear the attributes of a button of the button matrix

Parameters

- *btnm*: pointer to button matrix object
- *btn_id*: 0 based index of the button to modify. (Not counting new lines)

void **lv_btnm_set_btn_ctrl_all** (lv_obj_t **btnm*, lv_btnm_ctrl_t *ctrl*)
 Set the attributes of all buttons of a button matrix

Parameters

- *btnm*: pointer to a button matrix object
- *ctrl*: attribute(s) to set from lv_btnm_ctrl_t. Values can be ORed.

void **lv_btnm_clear_btn_ctrl_all** (lv_obj_t **btnm*, lv_btnm_ctrl_t *ctrl*)
 Clear the attributes of all buttons of a button matrix

Parameters

- *btnm*: pointer to a button matrix object
- *ctrl*: attribute(s) to set from lv_btnm_ctrl_t. Values can be ORed.
- *en*: true: set the attributes; false: clear the attributes

void **lv_btnm_set_btn_width** (**const** lv_obj_t **btnm*, uint16_t *btn_id*, uint8_t *width*)
 Set a single buttons relative width. This method will cause the matrix be regenerated and is a relatively expensive operation. It is recommended that initial width be specified using lv_btnm_set_ctrl_map and this method only be used for dynamic changes.

Parameters

- *btnm*: pointer to button matrix object
- *btn_id*: 0 based index of the button to modify.
- *width*: Relative width compared to the buttons in the same row. [1..7]

void **lv_btnm_set_one_toggle** (lv_obj_t **btnm*, bool *one_toggle*)
 Make the button matrix like a selector widget (only one button may be toggled at a time).

Toggling must be enabled on the buttons you want to be selected with lv_btnm_set_ctrl or lv_btnm_set_btn_ctrl_all.

Parameters

- *btnm*: Button matrix object
- *one_toggle*: Whether “one toggle” mode is enabled

const char ****lv_btnm_get_map_array** (**const** lv_obj_t **btnm*)
 Get the current map of a button matrix

Return the current map

Parameters

- `btnm`: pointer to a button matrix object

bool `lv_btnm_get_recolor` (const lv_obj_t **btnm*)

Check whether the button's text can use recolor or not

Return true: text recolor enable; false: disabled

Parameters

- `btnm`: pointer to button matrix object

uint16_t `lv_btnm_get_active_btn` (const lv_obj_t **btnm*)

Get the index of the lastly "activated" button by the user (pressed, released etc) Useful in the the `event_cb` to get the text of the button, check if hidden etc.

Return index of the last released button (LV_BTNM_BTN_NONE: if unset)

Parameters

- `btnm`: pointer to button matrix object

const char *`lv_btnm_get_active_btn_text` (const lv_obj_t **btnm*)

Get the text of the lastly "activated" button by the user (pressed, released etc) Useful in the the `event_cb`

Return text of the last released button (NULL: if unset)

Parameters

- `btnm`: pointer to button matrix object

uint16_t `lv_btnm_get_pressed_btn` (const lv_obj_t **btnm*)

Get the pressed button's index. The button be really pressed by the user or manually set to pressed with `lv_btnm_set_pressed`

Return index of the pressed button (LV_BTNM_BTN_NONE: if unset)

Parameters

- `btnm`: pointer to button matrix object

const char *`lv_btnm_get_btn_text` (const lv_obj_t **btnm*, uint16_t *btn_id*)

Get the button's text

Return text of `btn_index`' button

Parameters

- `btnm`: pointer to button matrix object
- `btn_id`: the index a button not counting new line characters. (The return value of `lv_btnm_get_pressed/released`)

bool `lv_btnm_get_btn_ctrl` (lv_obj_t **btnm*, uint16_t *btn_id*, lv_btnm_ctrl_t *ctrl*)

Get the whether a control value is enabled or disabled for button of a button matrix

Return true: long press repeat is disabled; false: long press repeat enabled

Parameters

- `btnm`: pointer to a button matrix object
- `btn_id`: the index a button not counting new line characters. (E.g. the return value of `lv_btnm_get_pressed/released`)
- `ctrl`: control values to check (ORed value can be used)

const lv_style_t ***lv_btmn_get_style** (**const** lv_obj_t **btmn*, lv_btmn_style_t *type*)

Get a style of a button matrix

Return style pointer to a style

Parameters

- *btmn*: pointer to a button matrix object
- *type*: which style should be get

bool **lv_btmn_get_one_toggle** (**const** lv_obj_t **btmn*)

Find whether “one toggle” mode is enabled.

Return whether “one toggle” mode is enabled

Parameters

- *btmn*: Button matrix object

struct lv_btmn_ext_t

#include <lv_btmn.h>

Public Members

const char ***map_p*

lv_area_t ***button_areas**

lv_btmn_ctrl_t ***ctrl_bits**

const lv_style_t ***styles_btn**[LV_BTN_STATE_NUM]

uint16_t **btn_cnt**

uint16_t **btn_id_pr**

uint16_t **btn_id_act**

uint8_t **recolor**

uint8_t **one_toggle**

5.6 Calendar (lv_calendar)

5.6.1 Overview

The Calendar object is a classic calendar which can:

- highlight the current day and week
- highlight any user-defined dates
- display the name of the days
- go the next/previous month by button click
- highlight the clicked day

The set and get dates in the calendar the lv_calendar_date_t type is used which is a structure with year, month and day fields.

To set the **current date** use the lv_calendar_set_today_date(*calendar*, &*today_date*) function.

To set the **shown date** use `lv_calendar_set_shown_date(calendar, &shown_date);`

The list of **highlighted dates** should be stored in a `lv_calendar_date_t` array a loaded by `lv_calendar_set_highlighted_dates(calendar, &highlighted_dates)`. Only the arrays pointer will be saved so the array should be a static or global variable.

The **name of the days** can be adjusted with `lv_calendar_set_day_names(calendar, day_names)` where `day_names` looks like `const char * day_names[7] = {"Su", "Mo", ...};`

5.6.2 Styles

You can set the styles with `lv_calendar_set_style(btn, LV_CALENDAR_STYLE_..., &style)`.

- **LV_CALENDAR_STYLE_BG** Style of the background using the body properties and the style of the date numbers using the text properties. `body.padding.left/right/bottom` will be added on the edges. around the date numbers.
- **LV_CALENDAR_STYLE_HEADER** Style of the header where the current year and month is displayed. body and text properties are used.
- **LV_CALENDAR_STYLE_HEADER_PR** Pressed header style, used when the next/prev. month button is being pressed. text properties are used by the arrows.
- **LV_CALENDAR_STYLE_DAY_NAMES** Style of the day names. text properties are used by the day texts and `body.padding.top` determines the space above the day names.
- **LV_CALENDAR_STYLE_HIGHLIGHTED_DAYS** text properties are used to adjust the style of the highlighted days
- **LV_CALENDAR_STYLE_INACTIVE_DAYS** text properties are used to adjust the style of the visible days of previous/next month.
- **LV_CALENDAR_STYLE_WEEK_BOX** body properties are used to set the style of the week box
- **LV_CALENDAR_STYLE_TODAY_BOX** body and text properties are used to set the style of the today box

5.6.3 Events

Besides the [Generic events](#) the following [Special events](#) are sent by the calendars: **LV_EVENT_VALUE_CHANGED** is sent when the current month has changed.

In Input device related events `lv_calendar_get_pressed_date(calendar)` tells which day is currently being pressed or return `NULL` if no date is pressed.

5.6.4 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.6.5 Example

C



Calendar image

code

```
static lv_style_t style_week_box;
lv_style_copy(&style_week_box, &lv_style_plain);
style_week_box.body.border.width = 1;
style_week_box.body.border.color = LV_COLOR_HEX3(0x333);
style_week_box.body.empty = 1;
style_week_box.body.radius = LV_RADIUS_CIRCLE;
style_week_box.body.padding.ver = 3;
style_week_box.body.padding.hor = 3;

/*Create a style for today*/
static lv_style_t style_today_box;
lv_style_copy(&style_today_box, &lv_style_plain);
style_today_box.body.border.width = 2;
style_today_box.body.border.color = LV_COLOR_NAVY;
style_today_box.body.empty = 1;
style_today_box.body.radius = LV_RADIUS_CIRCLE;
style_today_box.body.padding.ver = 3;
style_today_box.body.padding.hor = 3;
style_today_box.text.color= LV_COLOR_BLUE;

/*Create a style for the highlighted days*/
static lv_style_t style_highlighted_day;
lv_style_copy(&style_highlighted_day, &lv_style_plain);
style_highlighted_day.body.border.width = 2;
style_highlighted_day.body.border.color = LV_COLOR_NAVY;
style_highlighted_day.body.empty = 1;
style_highlighted_day.body.radius = LV_RADIUS_CIRCLE;
style_highlighted_day.body.padding.ver = 3;
style_highlighted_day.body.padding.hor = 3;
style_highlighted_day.text.color= LV_COLOR_BLUE;
```

(continues on next page)

(continued from previous page)

```

/*Apply the styles*/
lv_calendar_set_style(calendar, LV_CALENDAR_STYLE_WEEK_BOX, &style_week_box);
lv_calendar_set_style(calendar, LV_CALENDAR_STYLE_TODAY_BOX, &style_today_box);
lv_calendar_set_style(calendar, LV_CALENDAR_STYLE_HIGHLIGHTED_DAYS, &style_
↪highlighted_day);

/*Set the today*/
lv_calendar_date_t today;
today.year = 2018;
today.month = 10;
today.day = 23;

lv_calendar_set_today_date(calendar, &today);
lv_calendar_set_showed_date(calendar, &today);

/*Highlight some days*/
static lv_calendar_date_t highlihted_days[3];           /*Only it's pointer will be saved_
↪so should be static*/
highlihted_days[0].year = 2018;
highlihted_days[0].month = 10;
highlihted_days[0].day = 6;

highlihted_days[1].year = 2018;
highlihted_days[1].month = 10;
highlihted_days[1].day = 11;

highlihted_days[2].year = 2018;
highlihted_days[2].month = 11;
highlihted_days[2].day = 22;

lv_calendar_set_highlighted_dates(calendar, highlihted_days, 3);

```

MicroPython

No examples yet.

5.6.6 API

Typedefs

```
typedef uint8_t lv_calendar_style_t
```

Enums

```
enum [anonymous]
```

Values:

```
LV_CALENDAR_STYLE_BG
```

```
LV_CALENDAR_STYLE_HEADER
```

```
LV_CALENDAR_STYLE_HEADER_PR
```

`LV_CALENDAR_STYLE_DAY_NAMES`

`LV_CALENDAR_STYLE_HIGHLIGHTED_DAYS`

`LV_CALENDAR_STYLE_INACTIVE_DAYS`

`LV_CALENDAR_STYLE_WEEK_BOX`

`LV_CALENDAR_STYLE_TODAY_BOX`

Functions

`lv_obj_t *lv_calendar_create (lv_obj_t *par, const lv_obj_t *copy)`

Create a calendar objects Create a calendar object

Return pointer to the created calendar

Return pointer to the created calendar

Parameters

- `par`: pointer to an object, it will be the parent of the new calendar
- `copy`: pointer to a calendar object, if not NULL then the new object will be copied from it

Parameters

- `par`: pointer to an object, it will be the parent of the new calendar
- `copy`: pointer to a calendar object, if not NULL then the new object will be copied from it

`void lv_calendar_set_today_date (lv_obj_t *calendar, lv_calendar_date_t *today)`

Set the today's date

Parameters

- `calendar`: pointer to a calendar object
- `today`: pointer to an `lv_calendar_date_t` variable containing the date of today. The value will be saved it can be local variable too.

`void lv_calendar_set_showed_date (lv_obj_t *calendar, lv_calendar_date_t *showed)`

Set the currently showed

Parameters

- `calendar`: pointer to a calendar object
- `showed`: pointer to an `lv_calendar_date_t` variable containing the date to show. The value will be saved it can be local variable too.

`void lv_calendar_set_highlighted_dates (lv_obj_t *calendar, lv_calendar_date_t *highlighted, uint16_t date_num)`

Set the the highlighted dates

Parameters

- `calendar`: pointer to a calendar object
- `highlighted`: pointer to an `lv_calendar_date_t` array containing the dates. ONLY A POINTER WILL BE SAVED! CAN'T BE LOCAL ARRAY.
- `date_num`: number of dates in the array

`void lv_calendar_set_day_names (lv_obj_t *calendar, const char **day_names)`

Set the name of the days

Parameters

- `calendar`: pointer to a calendar object
- `day_names`: pointer to an array with the names. E.g. `const char * days[7] = {"Sun", "Mon", ...}` Only the pointer will be saved so this variable can't be local which will be destroyed later.

void **lv_calendar_set_month_names** (lv_obj_t **calendar*, const char ***day_names*)

Set the name of the month

Parameters

- `calendar`: pointer to a calendar object
- `day_names`: pointer to an array with the names. E.g. `const char * days[12] = {"Jan", "Feb", ...}` Only the pointer will be saved so this variable can't be local which will be destroyed later.

void **lv_calendar_set_style** (lv_obj_t **calendar*, lv_calendar_style_t *type*, const lv_style_t **style*)

Set a style of a calendar.

Parameters

- `calendar`: pointer to calendar object
- `type`: which style should be set
- `style`: pointer to a style

lv_calendar_date_t ***lv_calendar_get_today_date** (const lv_obj_t **calendar*)

Get the today's date

Return return pointer to an *lv_calendar_date_t* variable containing the date of today.

Parameters

- `calendar`: pointer to a calendar object

lv_calendar_date_t ***lv_calendar_get_showed_date** (const lv_obj_t **calendar*)

Get the currently showed

Return pointer to an *lv_calendar_date_t* variable containing the date is being shown.

Parameters

- `calendar`: pointer to a calendar object

lv_calendar_date_t ***lv_calendar_get_pressed_date** (const lv_obj_t **calendar*)

Get the the pressed date.

Return pointer to an *lv_calendar_date_t* variable containing the pressed date.

Parameters

- `calendar`: pointer to a calendar object

lv_calendar_date_t ***lv_calendar_get_highlighted_dates** (const lv_obj_t **calendar*)

Get the the highlighted dates

Return pointer to an *lv_calendar_date_t* array containing the dates.

Parameters

- `calendar`: pointer to a calendar object

uint16_t **lv_calendar_get_highlighted_dates_num** (const lv_obj_t **calendar*)

Get the number of the highlighted dates

Return number of highlighted days

Parameters

- `calendar`: pointer to a calendar object

const char **`lv_calendar_get_day_names` (**const** lv_obj_t **calendar*)

Get the name of the days

Return pointer to the array of day names

Parameters

- `calendar`: pointer to a calendar object

const char **`lv_calendar_get_month_names` (**const** lv_obj_t **calendar*)

Get the name of the month

Return pointer to the array of month names

Parameters

- `calendar`: pointer to a calendar object

const lv_style_t *`lv_calendar_get_style` (**const** lv_obj_t **calendar*, *lv_calendar_style_t* type)

Get style of a calendar.

Return style pointer to the style

Parameters

- `calendar`: pointer to calendar object
- `type`: which style should be get

struct `lv_calendar_date_t`

#include <lv_calendar.h>

Public Members

uint16_t **year**

int8_t **month**

int8_t **day**

struct `lv_calendar_ext_t`

#include <lv_calendar.h>

Public Members

lv_calendar_date_t **today**

lv_calendar_date_t **showed_date**

lv_calendar_date_t ***highlighted_dates**

uint8_t **highlighted_dates_num**

int8_t **btn_pressing**

lv_calendar_date_t **pressed_date**

const char ****day_names**

const char ****month_names**

```
const lv_style_t *style_header
const lv_style_t *style_header_pr
const lv_style_t *style_day_names
const lv_style_t *style_highlighted_days
const lv_style_t *style_inactive_days
const lv_style_t *style_week_box
const lv_style_t *style_today_box
```

5.7 Canvas (lv_canvas)

5.7.1 Overview

A canvas is like an image with a buffer where user can draw anything. To assign a buffer to a canvas use `lv_canvas_set_buffer(canvas, buffer, width, height, LV_IMG_CF_TRUE_COLOR_ALPHA)`. `buffer` is static buffer (not just a local variable) to hold the image of the canvas. For example `static lv_color_t buffer[LV_CANVAS_BUF_SIZE_TRUE_COLOR(width, height)]`. `LV_CANVAS_BUF_SIZE_...` macros help to determine the size of the buffer with different color formats.

To set a pixel on the canvas use `lv_canvas_set_px(canvas, x, y, LV_COLOR_RED)`. With `LV_IMG_CF_INDEXED_...` or `LV_IMG_CF_ALPHA_...` the index of the color or the alpha value needs to be passed as color. E.g. `lv_color_t c; c.full = 3;`

For `LV_IMG_CF_INDEXED_...` color formats the palette needs to be set with `lv_canvas_set_palette(canvas, 3, LV_COLOR_RED)`. It sets pixels with *index*=3 to red.

`lv_canvas_fill_bg(canvas, LV_COLOR_BLUE)` fills the whole canvas to blue.

An array of pixel can be copied to the canvas with `lv_canvas_copy_buf(canvas, buffer_to_copy, x, y, width, height)`. The color format of the buffer and the canvas need match.

To draw something to the canvas use

- `lv_canvas_draw_rect(canvas, x, y, width, height, &style)`
- `lv_canvas_draw_text(canvas, x, y, max_width, &style, txt, LV_LABEL_ALIGN_LEFT/CENTER/RIGHT)`
- `lv_canvas_draw_img(canvas, x, y, &img_src, &style)`
- `lv_canvas_draw_line(canvas, point_array, point_cnt, &style)`
- `lv_canvas_draw_polygon(canvas, points_array, point_cnt, &style)`
- `lv_canvas_draw_arc(canvas, x, y, radius, start_angle, end_angle, &style)`

An rotated image can be added to canvas with `lv_canvas_rotate(canvas, &img_dsc, angle, x, y, pivot_x, pivot_y)`. It will rotate the image shown by `img_dsc` around the given pivot and stores it on the `x, y` coordinates of canvas. Instead of `img_dsc` and the buffer of another canvas also can be used by `lv_canvas_get_img(canvas)`.

Note that a canvas can't be rotated on itself but a source and destination (the canvas).

5.7.2 Styles

You can set the styles with `lv_canvas_set_style(btn, LV_CANVAS_STYLE_MAIN, &style)`. `style.image.color` is used to tell the base color with `LV_IMG_CF_ALPHA...` color format.

5.7.3 Events

Only the [Generic events](#) are sent by the object type.

Learn more about [Events](#).

5.7.4 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.7.5 Example

C

code

MicroPython

No examples yet.

5.7.6 API

Defines

```
LV_CANVAS_BUF_SIZE_TRUE_COLOR(w, h)
LV_CANVAS_BUF_SIZE_TRUE_COLOR_CHROMA_KEYED(w, h)
LV_CANVAS_BUF_SIZE_TRUE_COLOR_ALPHA(w, h)
LV_CANVAS_BUF_SIZE_ALPHA_1BIT(w, h)
LV_CANVAS_BUF_SIZE_ALPHA_2BIT(w, h)
LV_CANVAS_BUF_SIZE_ALPHA_4BIT(w, h)
LV_CANVAS_BUF_SIZE_ALPHA_8BIT(w, h)
LV_CANVAS_BUF_SIZE_INDEXED_1BIT(w, h)
LV_CANVAS_BUF_SIZE_INDEXED_2BIT(w, h)
LV_CANVAS_BUF_SIZE_INDEXED_4BIT(w, h)
LV_CANVAS_BUF_SIZE_INDEXED_8BIT(w, h)
```

Typedefs

```
typedef uint8_t lv_canvas_style_t
```

Enums

```
enum [anonymous]
```

Values:

```
LV_CANVAS_STYLE_MAIN
```

Functions

```
lv_obj_t *lv_canvas_create(lv_obj_t *par, const lv_obj_t *copy)
```

Create a canvas object

Return pointer to the created canvas

Parameters

- `par`: pointer to an object, it will be the parent of the new canvas
- `copy`: pointer to a canvas object, if not NULL then the new object will be copied from it

```
void lv_canvas_set_buffer(lv_obj_t *canvas, void *buf, lv_coord_t w, lv_coord_t h, lv_img_cf_t cf)
```

Set a buffer for the canvas. Set a buffer for the canvas.

Parameters

- `buf`: a buffer where the content of the canvas will be. The required size is $(lv_img_color_format_get_px_size(cf) * w * h) / 8$ It can be allocated with `lv_mem_alloc()` or it can be statically allocated array (e.g. `static lv_color_t buf[100*50]`) or it can be an address in RAM or external SRAM
- `canvas`: pointer to a canvas object
- `w`: width of the canvas
- `h`: height of the canvas
- `cf`: color format. `LV_IMG_CF_...`

Parameters

- `buf`: a buffer where the content of the canvas will be. The required size is $(lv_img_color_format_get_px_size(cf) * w * h) / 8$ It can be allocated with `lv_mem_alloc()` or it can be statically allocated array (e.g. `static lv_color_t buf[100*50]`) or it can be an address in RAM or external SRAM
- `canvas`: pointer to a canvas object
- `w`: width of the canvas
- `h`: height of the canvas
- `cf`: color format. The following formats are supported: `LV_IMG_CF_TRUE_COLOR`, `LV_IMG_CF_TRUE_COLOR_CHROMA_KEYED`, `LV_IMG_CF_INDEXES_1/2/4/8BIT`

```
void lv_canvas_set_px(lv_obj_t *canvas, lv_coord_t x, lv_coord_t y, lv_color_t c)
```

Set the color of a pixel on the canvas Set the color of a pixel on the canvas

Parameters

- canvas:
- x: x coordinate of the point to set
- y: x coordinate of the point to set
- c: color of the point

Parameters

- canvas: pointer to canvas object
- x: x coordinate of the point to set
- y: x coordinate of the point to set
- c: color of the point

void **lv_canvas_set_palette** (lv_obj_t **canvas*, uint8_t *id*, lv_color_t *c*)

Set the palette color of a canvas with index format. Valid only for LV_IMG_CF_INDEXED1/2/4/8

Parameters

- canvas: pointer to canvas object
- id: the palette color to set:
 - for LV_IMG_CF_INDEXED1: 0..1
 - for LV_IMG_CF_INDEXED2: 0..3
 - for LV_IMG_CF_INDEXED4: 0..15
 - for LV_IMG_CF_INDEXED8: 0..255
- c: the color to set

void **lv_canvas_set_style** (lv_obj_t **canvas*, *lv_canvas_style_t* type, const lv_style_t **style*)

Set a style of a canvas.

Parameters

- canvas: pointer to canvas object
- type: which style should be set
- style: pointer to a style

lv_color_t **lv_canvas_get_px** (lv_obj_t **canvas*, lv_coord_t *x*, lv_coord_t *y*)

Get the color of a pixel on the canvas

Return color of the point

Parameters

- canvas:
- x: x coordinate of the point to set
- y: x coordinate of the point to set

lv_img_dsc_t ***lv_canvas_get_img** (lv_obj_t **canvas*)

Get the image of the canvas as a pointer to an lv_img_dsc_t variable.

Return pointer to the image descriptor.

Parameters

- canvas: pointer to a canvas object

const lv_style_t *lv_canvas_get_style (const lv_obj_t *canvas, lv_canvas_style_t type)

Get style of a canvas.

Return style pointer to the style

Parameters

- canvas: pointer to canvas object
- type: which style should be get

void **lv_canvas_copy_buf** (lv_obj_t *canvas, const void *to_copy, lv_coord_t x, lv_coord_t y, lv_coord_t w, lv_coord_t h)

Copy a buffer to the canvas Copy a buffer to the canvas

Parameters

- canvas: pointer to a canvas object
- to_copy: buffer to copy. The color format has to match with the canvas's buffer color format
- x: left side of the destination position
- y: top side of the destination position
- w: width of the buffer to copy
- h: height of the buffer to copy

Parameters

- canvas: pointer to a canvas object
- to_copy: buffer to copy. The color format has to match with the canvas's buffer color format
- w: width of the buffer to copy
- h: height of the buffer to copy
- x: left side of the destination position
- y: top side of the destination position

void **lv_canvas_rotate** (lv_obj_t *canvas, lv_img_dsc_t *img, int16_t angle, lv_coord_t offset_x, lv_coord_t offset_y, int32_t pivot_x, int32_t pivot_y)

Rotate and image and store the result on a canvas.

Parameters

- canvas: pointer to a canvas object
- img: pointer to an image descriptor. Can be the image descriptor of an other canvas too (*lv_canvas_get_img()*).
- angle: the angle of rotation (0..360);
- offset_x: offset X to tell where to put the result data on destination canvas
- offset_y: offset Y to tell where to put the result data on destination canvas
- pivot_x: pivot X of rotation. Relative to the source canvas Set to source width / 2 to rotate around the center
- pivot_y: pivot Y of rotation. Relative to the source canvas Set to source height / 2 to rotate around the center

void **lv_canvas_fill_bg** (lv_obj_t *canvas, lv_color_t color)

Fill the canvas with color

Parameters

- `canvas`: pointer to a canvas
- `color`: the background color

```
void lv_canvas_draw_rect(lv_obj_t *canvas, lv_coord_t x, lv_coord_t y, lv_coord_t w, lv_coord_t h,  
                        const lv_style_t *style)
```

Draw a rectangle on the canvas

Parameters

- canvas: pointer to a canvas object
- x: left coordinate of the rectangle
- y: top coordinate of the rectangle
- w: width of the rectangle
- h: height of the rectangle
- style: style of the rectangle (body properties are used except padding)

```
void lv_canvas_draw_text (lv_obj_t *canvas, lv_coord_t x, lv_coord_t y, lv_coord_t max_w, const
                        lv_style_t *style, const char *txt, lv_label_align_t align)
```

Draw a text on the canvas.

Parameters

- **canvas**: pointer to a canvas object
- **x**: left coordinate of the text
- **y**: top coordinate of the text
- **max_w**: max width of the text. The text will be wrapped to fit into this size
- **style**: style of the text (text properties are used)
- **txt**: text to display
- **align**: align of the text (LV_LABEL_ALIGN_LEFT/RIGHT/CENTER)

```
void lv_canvas_draw_img (lv_obj_t *canvas, lv_coord_t x, lv_coord_t y, const void *src, const
lv_style_t *style)
```

Draw an image on the canvas

Parameters

- `canvas`: pointer to a canvas object
- `src`: image source. Can be a pointer an `lv_img_dsc_t` variable or a path an image.
- `style`: style of the image (image properties are used)

```
void lv_canvas_draw_line (lv_obj_t *canvas, const lv_point_t *points, uint32_t point_cnt, const
                        lv_style_t *style)
```

Draw a line on the canvas

Parameters

- `canvas`: pointer to a canvas object
- `points`: point of the line
- `point_cnt`: number of points
- `style`: style of the line (line properties are used)

```
void lv_canvas_draw_polygon(lv_obj_t *canvas, const lv_point_t *points, uint32_t point_cnt,
                           const lv_style_t *style)
```

Draw a polygon on the canvas

Parameters

- canvas: pointer to a canvas object
- points: point of the polygon
- point_cnt: number of points
- style: style of the polygon (body.main_color and body.opa is used)

```
void lv_canvas_draw_arc(lv_obj_t *canvas, lv_coord_t x, lv_coord_t y, lv_coord_t r, int32_t
                       start_angle, int32_t end_angle, const lv_style_t *style)
```

Draw an arc on the canvas

Parameters

- canvas: pointer to a canvas object
- x: origo x of the arc
- y: origo y of the arc
- r: radius of the arc
- start_angle: start angle in degrees
- end_angle: end angle in degrees
- style: style of the polygon (body.main_color and body.opa is used)

```
struct lv_canvas_ext_t
#include <lv_canvas.h>
```

Public Members

lv_img_ext_t **img**

lv_img_dsc_t **dsc**

5.8 Check box (lv_cb)

5.8.1 Overview

The Check Box objects are built from a Button **background** which contains an also Button **bullet** and a **label** to realize a classical check box. The **text** can be modified by the `lv_cb_set_text(cb, "New text")` function. It will dynamically allocate the text. To set a static text use `lv_cb_set_static_text(cb, txt_buf)`. This way only a pointer will be stored to `txt_buf` so it needs shouldn't deallocated while the checkbox exists.

You can manually **check** / **un-check** the Check box via `lv_cb_set_checked(cb, true/false)`.

To make the checkbox inactive use `lv_cb_set_inactive(cb, true)`.

5.8.2 Styles

The Check box styles can be modified with `lv_cb_set_style(cb, LV_CB_STYLE_..., &style)`.

- **LV_CB_STYLE_BG** Background style. Uses all `style.body` properties. The label's style comes from `style.text`. Default: `lv_style_transp`
- **LV_CB_STYLE_BOX_REL** Style of the released box. Uses the `style.body` properties. Default: `lv_style_btn_rel`
- **LV_CB_STYLE_BOX_PR** Style of the pressed box. Uses the `style.body` properties. Default: `lv_style_btn_pr`
- **LV_CB_STYLE_BOX_TGL_REL** Style of the checked released box. Uses the `style.body` properties. Default: `lv_style_btn_tgl_rel`
- **LV_CB_STYLE_BOX_TGL_PR** Style of the checked released box. Uses the `style.body` properties. Default: `lv_style_btn_tgl_pr`
- **LV_CB_STYLE_BOX_INA** Style of the inactive box. Uses the `style.body` properties. Default: `lv_style_btn_ina`

5.8.3 Events

Besided the [Generic events](#) the following [Special events](#) are sent by the Check boxes:

- **LV_EVENT_VALUE_CHANGED** sent when the Check box is toggled.

Note that the generic input device related events (like `LV_EVENT_PRESSED`) are sent in inactive state too. You need to check the state with `lv_cb_is_inactive(cb)` to ignore the events from inactive Check boxes.

Learn more about [Events](#).

5.8.4 Keys

The following *Keys* are processed by the Buttons:

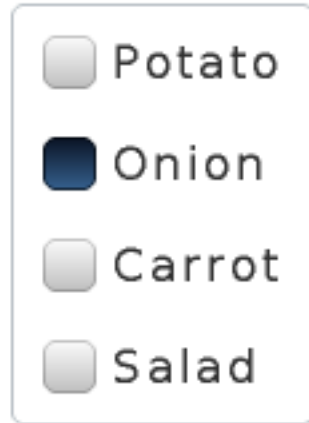
- **LV_KEY_RIGHT/UP** Go to toggled state if toggling is enabled
- **LV_KEY_LEFT/DOWN** Go to non-toggled state if toggling is enabled

Note that, as usual, the state of `LV_KEY_ENTER` is translated to `LV_EVENT_PRESSED/PRESSING/RELEASED` etc.

Learn more about [Keys](#).

5.8.5 Example

C



Checkbox image

code

```
static lv_res_t cb_release_action(lv_obj_t * cb)
{
    /*A check box is clicked*/
    printf("%s state: %d\n", lv_cb_get_text(cb), lv_cb_is_checked(cb));

    return LV_RES_OK;
}

.
.
.

/*****
 * Create a container for the check boxes
 *****/

/*Create border style*/
static lv_style_t style_border;
lv_style_copy(&style_border, &lv_style_pretty_color);
style_border.glass = 1;
style_border.body.empty = 1;

/*Create a container*/
lv_obj_t * cont;
cont = lv_cont_create(lv_scr_act(), NULL);
lv_cont_set_layout(cont, LV_LAYOUT_COL_L);
lv_cont_set_fit(cont, true, true);
lv_obj_set_style(cont, &style_border);

/*****/
```

(continues on next page)

(continued from previous page)

```
* Create check boxes
*****

/*Create check box*/
lv_obj_t * cb;
cb = lv_cb_create(cont, NULL);
lv_cb_set_text(cb, "Potato");
lv_cb_set_action(cb, cb_release_action);

/*Copy the previous check box*/
cb = lv_cb_create(cont, cb);
lv_cb_set_text(cb, "Onion");

/*Copy the previous check box*/
cb = lv_cb_create(cont, cb);
lv_cb_set_text(cb, "Carrot");

/*Copy the previous check box*/
cb = lv_cb_create(cont, cb);
lv_cb_set_text(cb, "Salad");

/*Align the container to the middle*/
lv_obj_align(cont, NULL, LV_ALIGN_CENTER, 0, 0);
```

MicroPython

No examples yet.

5.8.6 API

Typedefs

```
typedef uint8_t lv_cb_style_t
```

Enums

```
enum [anonymous]
```

Values:

```
LV_CB_STYLE_BG
LV_CB_STYLE_BOX_REL
LV_CB_STYLE_BOX_PR
LV_CB_STYLE_BOX_TGL_REL
LV_CB_STYLE_BOX_TGL_PR
LV_CB_STYLE_BOX_INA
```

Functions

`lv_obj_t *lv_cb_create (lv_obj_t *par, const lv_obj_t *copy)`

Create a check box objects

Return pointer to the created check box

Parameters

- `par`: pointer to an object, it will be the parent of the new check box
- `copy`: pointer to a check box object, if not NULL then the new object will be copied from it

void `lv_cb_set_text (lv_obj_t *cb, const char *txt)`

Set the text of a check box. `txt` will be copied and may be deallocated after this function returns.

Parameters

- `cb`: pointer to a check box
- `txt`: the text of the check box. NULL to refresh with the current text.

void `lv_cb_set_static_text (lv_obj_t *cb, const char *txt)`

Set the text of a check box. `txt` must not be deallocated during the life of this checkbox.

Parameters

- `cb`: pointer to a check box
- `txt`: the text of the check box. NULL to refresh with the current text.

static void `lv_cb_set_checked (lv_obj_t *cb, bool checked)`

Set the state of the check box

Parameters

- `cb`: pointer to a check box object
- `checked`: true: make the check box checked; false: make it unchecked

static void `lv_cb_set_inactive (lv_obj_t *cb)`

Make the check box inactive (disabled)

Parameters

- `cb`: pointer to a check box object

void `lv_cb_set_style (lv_obj_t *cb, lv_cb_style_t type, const lv_style_t *style)`

Set a style of a check box

Parameters

- `cb`: pointer to check box object
- `type`: which style should be set
- `style`: pointer to a style

const char *`lv_cb_get_text (const lv_obj_t *cb)`

Get the text of a check box

Return pointer to the text of the check box

Parameters

- `cb`: pointer to check box object

static bool **lv_cb_is_checked**(const lv_obj_t *cb)

Get the current state of the check box

Return true: checked; false: not checked

Parameters

- cb: pointer to a check box object

static bool **lv_cb_is_inactive**(const lv_obj_t *cb)

Get whether the check box is inactive or not.

Return true: inactive; false: not inactive

Parameters

- cb: pointer to a check box object

const lv_style_t ***lv_cb_get_style**(const lv_obj_t *cb, lv_cb_style_t type)

Get a style of a button

Return style pointer to the style

Parameters

- cb: pointer to check box object
- type: which style should be get

struct lv_cb_ext_t

#include <lv_cb.h>

Public Members

lv_btn_ext_t **bg_btn**

lv_obj_t ***bullet**

lv_obj_t ***label**

5.9 Chart (lv_chart)

5.9.1 Overview

Charts have a rectangle-like background with horizontal and vertical division lines. You can add any number of **series** to the charts by `lv_chart_add_series(chart, color)`. It allocates data for a `lv_chart_series_t` structure which contains the chosen `color` and an array for the data points.

You have several options to set the data of series:

1. Set the values manually in the array like `ser1->points[3] = 7` and refresh the chart with `lv_chart_refresh(chart)`.
2. Use the `lv_chart_set_next(chart, ser, value)`
3. Initialize all points to a given value with: `lv_chart_init_points(chart, ser, value)`.
4. Set all points from an array with: `lv_chart_set_points(chart, ser, value_array)`.

`lv_chart_set_next` can behave in two way depending on *update mode*:

- **LV_CHART_UPDATE_MODE_SHIFT** Shift old data to the left and add the new one o the right

- **LV_CHART_UPDATE_MODE_CIRCULAR** Add the new data in a circular way. (Like an ECG diagram)

To update mode can be changed with `lv_chart_set_update_mode(chart, LV_CHART_UPDATE_MODE_...)`.

The following **data display types** exists:

- **LV_CHART_TYPE_NONE** do not display any data. It can be used to hide a serie.
- **LV_CHART_TYPE_LINE** draw lines between the points
- **LV_CHART_TYPE_COL** Draw columns
- **LV_CHART_TYPE_POINT** Draw points
- **LV_CHART_TYPE_AREA** Draw areas (fill the area below the lines)
- **LV_CHART_TYPE_VERTICAL_LINE** Draw only vertical lines to connect the points. Useful if the chart width is equal to the number of points.

You can specify the display type with `lv_chart_set_type(chart, LV_CHART_TYPE_...)`. The types can be ORed (like `LV_CHART_TYPE_LINE | LV_CHART_TYPE_POINT`).

You can specify a the **min. and max. values in y** directions with `lv_chart_set_range(chart, y_min, y_max)`. The value of the points will be scaled proportionally. The default range is: 0..100.

The **number of points** in the data lines can be modified by `lv_chart_set_point_count(chart, point_num)`. The default value is 10.

The **number of horizontal and vertical division lines** can be modified by `lv_chart_set_div_line_count(chart, hdiv_num, vdiv_num)`. The default settings are 3 horizontal and 5 vertical division lines.

To set the **line width** and **point radius** of the series use the `lv_chart_set_series_width(chart, size)` function. The default value is: 2.

The ***opacity of the data lines** can be specified by `lv_chart_set_series_opa(chart, opa)`. The default value is: OPA_COVER.

You can apply a **dark color fade** on the bottom of columns and points by `lv_chart_set_series_darking(chart, effect)` function. The default dark level is OPA_50.

Ticks and texts to ticks can be added with

```
lv_chart_set_x_ticks(chart, list_of_values, num_tick_marks,
                    major_tick_len, minor_tick_len,
                    LV_CHART_AXIS_DRAW_LAST_TICK);
```

`list_of_values` is an array with `num_tick_marks` '\n' terminated text (except the last) with text for the ticks. E.g. `const char * list_of_values = "first\nsecond\nthird"`. `major_tick_len` and `minor_tick_len` is the length of the tick marks when the tick is on the division line or when it isn't respectively. `LV_CHART_AXIS_DRAW_LAST_TICK` is the only supported mode now.

For y axis you can use `lv_chart_set_y_ticks`.

`lv_chart_set_margin(chart, 20)` needs to be used to add some extra space around the chart for the ticks and texts.

5.9.2 Styles

You can set the styles with `lv_chart_set_style(btn, LV_CHART_STYLE_MAIN, &style)`.

- **style.body** properties set the background's appearance

- **style.line** properties set the division lines' appearance
- **style.text** properties set the axis labels' appearance

5.9.3 Events

Only the [Generic events](#) are sent by the object type.

Learn more about [Events](#).

5.9.4 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.9.5 Example

C

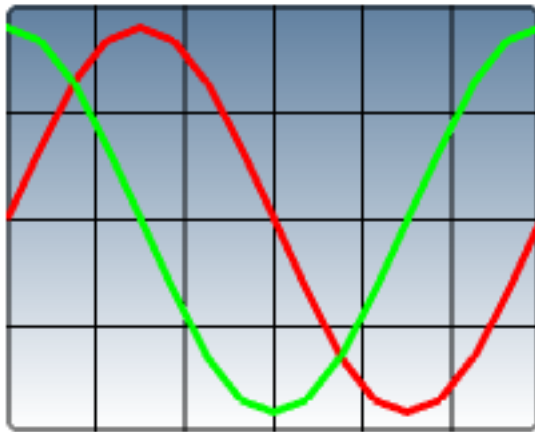


Chart image

code

```
/*Create a style for the chart*/
static lv_style_t style;
lv_style_copy(&style, &lv_style_pretty);
style.body.shadow.width = 6;
style.body.shadow.color = LV_COLOR_GRAY;
style.line.color = LV_COLOR_GRAY;

/*Create a chart*/
lv_obj_t * chart;
chart = lv_chart_create(lv_scr_act(), NULL);
```

(continues on next page)

(continued from previous page)

```
lv_obj_set_size(chart, 200, 150);
lv_obj_set_style(chart, &style);
lv_obj_align(chart, NULL, LV_ALIGN_CENTER, 0, 0);
lv_chart_set_type(chart, LV_CHART_TYPE_POINT | LV_CHART_TYPE_LINE);  /*Show lines_
↪and points too*/
lv_chart_set_series_opa(chart, LV_OPA_70);                             /*Opacity of_
↪the data series*/
lv_chart_set_series_width(chart, 4);                                   /*Line width_
↪and point radius*/

lv_chart_set_range(chart, 0, 100);

/*Add two data series*/
lv_chart_series_t * ser1 = lv_chart_add_series(chart, LV_COLOR_RED);
lv_chart_series_t * ser2 = lv_chart_add_series(chart, LV_COLOR_GREEN);

/*Set the next points on 'dl1'*/
lv_chart_set_next(chart, ser1, 10);
lv_chart_set_next(chart, ser1, 50);
lv_chart_set_next(chart, ser1, 70);
lv_chart_set_next(chart, ser1, 90);

/*Directly set points on 'dl2'*/
ser2->points[0] = 90;
ser2->points[1] = 70;
ser2->points[2] = 65;
ser2->points[3] = 65;
ser2->points[4] = 65;
ser2->points[5] = 65;

lv_chart_refresh(chart); /*Required after direct set*/
```

MicroPython

No examples yet.

5.9.6 API

Defines

LV_CHART_POINT_DEF

Typedefs

```
typedef uint8_t lv_chart_type_t
typedef uint8_t lv_chart_update_mode_t
typedef uint8_t lv_chart_axis_options_t
typedef uint8_t lv_chart_style_t
```

Enums

enum [anonymous]

Values:

```
LV_CHART_TYPE_NONE = 0x00
LV_CHART_TYPE_LINE = 0x01
LV_CHART_TYPE_COLUMN = 0x02
LV_CHART_TYPE_POINT = 0x04
LV_CHART_TYPE_VERTICAL_LINE = 0x08
LV_CHART_TYPE_AREA = 0x10
```

enum [anonymous]

Values:

```
LV_CHART_UPDATE_MODE_SHIFT
LV_CHART_UPDATE_MODE_CIRCULAR
```

enum [anonymous]

Values:

```
LV_CHART_AXIS_DRAW_LAST_TICK = 0x01
```

enum [anonymous]

Values:

```
LV_CHART_STYLE_MAIN
```

Functions

`lv_obj_t *lv_chart_create(lv_obj_t *par, const lv_obj_t *copy)`

Create a chart background objects

Return pointer to the created chart background

Parameters

- `par`: pointer to an object, it will be the parent of the new chart background
- `copy`: pointer to a chart background object, if not NULL then the new object will be copied from it

`lv_chart_series_t *lv_chart_add_series(lv_obj_t *chart, lv_color_t color)`

Allocate and add a data series to the chart

Return pointer to the allocated data series

Parameters

- `chart`: pointer to a chart object
- `color`: color of the data series

`void lv_chart_clear_series(lv_obj_t *chart, lv_chart_series_t *serie)`

Clear the point of a serie

Parameters

- `chart`: pointer to a chart object
- `serie`: pointer to the chart's serie to clear

void **lv_chart_set_div_line_count** (lv_obj_t **chart*, uint8_t *hdiv*, uint8_t *vdiv*)
 Set the number of horizontal and vertical division lines

Parameters

- *chart*: pointer to a graph background object
- *hdiv*: number of horizontal division lines
- *vdiv*: number of vertical division lines

void **lv_chart_set_range** (lv_obj_t **chart*, lv_coord_t *ymin*, lv_coord_t *ymax*)
 Set the minimal and maximal y values

Parameters

- *chart*: pointer to a graph background object
- *ymin*: y minimum value
- *ymax*: y maximum value

void **lv_chart_set_type** (lv_obj_t **chart*, *lv_chart_type_t* *type*)
 Set a new type for a chart

Parameters

- *chart*: pointer to a chart object
- *type*: new type of the chart (from 'lv_chart_type_t' enum)

void **lv_chart_set_point_count** (lv_obj_t **chart*, uint16_t *point_cnt*)
 Set the number of points on a data line on a chart

Parameters

- *chart*: pointer to chart object
- *point_cnt*: new number of points on the data lines

void **lv_chart_set_series_opa** (lv_obj_t **chart*, lv_opa_t *opa*)
 Set the opacity of the data series

Parameters

- *chart*: pointer to a chart object
- *opa*: opacity of the data series

void **lv_chart_set_series_width** (lv_obj_t **chart*, lv_coord_t *width*)
 Set the line width or point radius of the data series

Parameters

- *chart*: pointer to a chart object
- *width*: the new width

void **lv_chart_set_series_darking** (lv_obj_t **chart*, lv_opa_t *dark_eff*)
 Set the dark effect on the bottom of the points or columns

Parameters

- *chart*: pointer to a chart object
- *dark_eff*: dark effect level (LV_OPA_TRANSP to turn off)

void **lv_chart_init_points** (lv_obj_t **chart*, *lv_chart_series_t* **ser*, lv_coord_t *y*)
 Initialize all data points with a value

Parameters

- `chart`: pointer to chart object
- `ser`: pointer to a data series on ‘chart’
- `y`: the new value for all points

void **lv_chart_set_points** (lv_obj_t **chart*, *lv_chart_series_t* **ser*, lv_coord_t *y_array*[])
Set the value of points from an array

Parameters

- `chart`: pointer to chart object
- `ser`: pointer to a data series on ‘chart’
- `y_array`: array of ‘lv_coord_t’ points (with ‘points count’ elements)

void **lv_chart_set_next** (lv_obj_t **chart*, *lv_chart_series_t* **ser*, lv_coord_t *y*)
Shift all data right and set the most right data on a data line Shift all data left and set the rightmost data on a data line

Parameters

- `chart`: pointer to chart object
- `ser`: pointer to a data series on ‘chart’
- `y`: the new value of the most right data

Parameters

- `chart`: pointer to chart object
- `ser`: pointer to a data series on ‘chart’
- `y`: the new value of the rightmost data

void **lv_chart_set_update_mode** (lv_obj_t **chart*, *lv_chart_update_mode_t* *update_mode*)
Set update mode of the chart object.

Parameters

- `chart`: pointer to a chart object
- `update`: mode

static void **lv_chart_set_style** (lv_obj_t **chart*, *lv_chart_style_t* *type*, **const** lv_style_t **style*)
Set the style of a chart

Parameters

- `chart`: pointer to a chart object
- `type`: which style should be set (can be only LV_CHART_STYLE_MAIN)
- `style`: pointer to a style

void **lv_chart_set_margin** (lv_obj_t **chart*, uint16_t *margin*)
Set the margin around the chart, used for axes value and labels Set the margin around the chart, used for axes value and labels

Parameters

- `chart`: pointer to an chart object
- `margin`: value of the margin

Parameters

- margin: value of the margin

void **lv_chart_set_x_ticks** (lv_obj_t *chart, const char *list_of_values, uint8_t num_tick_marks, uint8_t major_tick_len, uint8_t minor_tick_len, lv_chart_axis_options_t options)

Set the y-axis ticks of a chart Set the x/y-axis ticks of a chart

Parameters

- chart: pointer to a chart object
- list_of_values: list of string values, terminated with , except the last
- num_tick_marks: if list_of_values is NULL: total number of ticks per axis else step in ticks between two value labels
- major_tick_len: the length of the major tick, AUTO if 0
- minor_tick_len: the length of the minor tick, AUTO if 0
- options: extra options

Parameters

- chart: pointer to a chart object
- list_of_values: list of string values, terminated with , except the last
- num_tick_marks: if list_of_values is NULL: total number of ticks per axis else step in ticks between two value labels
- major_tick_len: the length of the major tick, AUTO if 0
- minor_tick_len: the length of the minor tick, AUTO if 0
- options: extra options

void **lv_chart_set_y_ticks** (lv_obj_t *chart, const char *list_of_values, uint8_t num_tick_marks, uint8_t major_tick_len, uint8_t minor_tick_len, lv_chart_axis_options_t options)

Set the y-axis ticks of a chart

Parameters

- chart: pointer to a chart object
- list_of_values: list of string values, terminated with , except the last
- num_tick_marks: if list_of_values is NULL: total number of ticks per axis else step in ticks between two value labels
- major_tick_len: the length of the major tick, AUTO if 0
- minor_tick_len: the length of the minor tick, AUTO if 0
- options: extra options

lv_chart_type_t **lv_chart_get_type** (const lv_obj_t *chart)

Get the type of a chart

Return type of the chart (from 'lv_chart_t' enum)

Parameters

- chart: pointer to chart object

`uint16_t lv_chart_get_point_cnt (const lv_obj_t *chart)`

Get the data point number per data line on chart

Return point number on each data line

Parameters

- `chart`: pointer to chart object

`lv_opa_t lv_chart_get_series_opa (const lv_obj_t *chart)`

Get the opacity of the data series

Return the opacity of the data series

Parameters

- `chart`: pointer to chart object

`lv_coord_t lv_chart_get_series_width (const lv_obj_t *chart)`

Get the data series width

Return the width the data series (lines or points)

Parameters

- `chart`: pointer to chart object

`lv_opa_t lv_chart_get_series_darking (const lv_obj_t *chart)`

Get the dark effect level on the bottom of the points or columns

Return dark effect level (LV_OPA_TRANSP to turn off)

Parameters

- `chart`: pointer to chart object

`static const lv_style_t *lv_chart_get_style (const lv_obj_t *chart, lv_chart_style_t type)`

Get the style of an chart object

Return pointer to the chart's style

Parameters

- `chart`: pointer to an chart object
- `type`: which style should be get (can be only LV_CHART_STYLE_MAIN)

`uint16_t lv_chart_get_margin (lv_obj_t *chart)`

Get the margin around the chart, used for axes value and labels

Parameters

- `chart`: pointer to an chart object
- `return`: value of the margin

`void lv_chart_refresh (lv_obj_t *chart)`

Refresh a chart if its data line has changed

Parameters

- `chart`: pointer to chart object

`struct lv_chart_series_t`

`#include <lv_chart.h>`

Public Members

```
lv_coord_t *points
lv_color_t color
uint16_t start_point
struct lv_chart_axis_cfg_t
    #include <lv_chart.h>
```

Public Members

```
const char *list_of_values
uint8_t num_tick_marks
lv_chart_axis_options_t options
uint8_t major_tick_len
uint8_t minor_tick_len
struct lv_chart_ext_t
    #include <lv_chart.h>
```

Public Members

```
lv_ll_t series_ll
lv_coord_t ymin
lv_coord_t ymax
uint8_t hdiv_cnt
uint8_t vdiv_cnt
uint16_t point_cnt
lv_chart_type_t type
lv_chart_axis_cfg_t y_axis
lv_chart_axis_cfg_t x_axis
uint16_t margin
uint8_t update_mode
lv_coord_t width
uint8_t num
lv_opa_t opa
lv_opa_t dark
struct lv_chart_ext_t::[anonymous] series
```

5.10 Container

5.10.1 Overview

The containers are **rectangle-like object** with some special features.

You can apply a **layout** on the containers to automatically order their children. The layout spacing comes from `style.body.padding. . .` properties. The possible layout options:

- **LV_LAYOUT_OFF** Do not align the children
- **LV_LAYOUT_CENTER** Align children to the center in column and keep `padding.inner` space between them
- **LV_LAYOUT_COL_**: Align children in a left justified column. Keep `padding.left` space on the left, `padding.top` space on the top and `padding.inner` space between the children.
- **LV_LAYOUT_COL_M** Align children in centered column. Keep `padding.top` space on the top and `padding.inner` space between the children.
- **LV_LAYOUT_COL_R** Align children in a right justified column. Keep `padding.right` space on the right, `padding.top` space on the top and `padding.inner` space between the children.
- **LV_LAYOUT_ROW_T** Align children in a top justified row. Keep `padding.left` space on the left, `padding.top` space on the top and `padding.inner` space between the children.
- **LV_LAYOUT_ROW_M** Align children in centered row. Keep `padding.left` space on the left and `padding.inner` space between the children.
- **LV_LAYOUT_ROW_B** Align children in a bottom justified row. Keep `padding.left` space on the left, `padding.bottom` space on the bottom and `padding.inner` space between the children.
- **LV_LAYOUT_PRETTY** Put as may objects as possible in a row (with at least `padding.inner` space and `padding.left/right` space on the sides). Divide the space in each line equally between the children. Keep `padding.top` space on the top and `padding.inner` space between the lines.
- **LV_LAYOUT_GRID** Similar to **LV_LAYOUT_PRETTY** but not divide horizontal space equally just let `padding.left/right` on the edges and `padding.inner` space between the elements.

Containers have an **auto fit** feature which can automatically change the size of the container according to its children and/or parent. The following options exist:

- **LV_FIT_NONE** Do not change the size automatically
- **LV_FIT_TIGHT** Set the size to involve all children by keeping `padding.top/bottom/left/right` space on the edges.
- **LV_FIT_FLOOD** Set the size to the parent's size by keeping `padding.top/bottom/left/right` (from the parent's style) space.
- **LV_FIT_FILL** Use **LV_FIT_FLOOD** while smaller than the parent and **LV_FIT_TIGHT** when larger.

To set the auto fit use `lv_cont_set_fit(cont, LV_FIT_...)`. It will set the same auto fit in every direction. To use different auto fit horizontally and vertically use `lv_cont_set_fit2(cont, hor_fit_type, ver_fit_type)`. To use different auto fit in all 4 directions use `lv_cont_set_fit4(cont, left_fit_type, right_fit_type, top_fit_type, bottom_fit_type)`.

5.10.2 Styles

You can set the styles with `lv_cont_set_style(btn, LV_CONT_STYLE_MAIN, &style)`.

- `style.body` properties are used.

5.10.3 Events

Only the [Generic events](#) are sent by the object type.

Learn more about [Events](#).

5.10.4 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.10.5 Keys

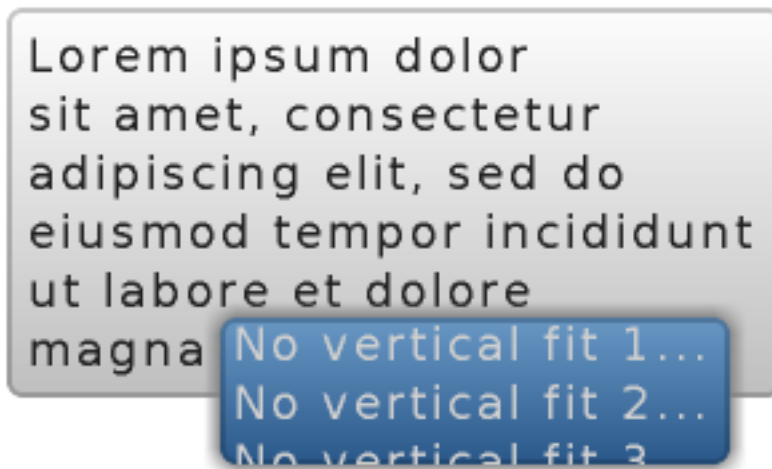
The following *Keys* are processed by the Buttons:

- **LV_KEY_RIGHT/UP** Go to toggled state if toggling is enabled
- **LV_KEY_LEFT/DOWN** Go to non-toggled state if toggling is enabled

Note that, as usual, the state of `LV_KEY_ENTER` is translated to `LV_EVENT_PRESSED/PRESSING/RELEASED` etc.

5.10.6 Example

C



Container image

code

```
lv_obj_t * box1;
box1 = lv_cont_create(lv_scr_act(), NULL);
lv_obj_set_style(box1, &lv_style_pretty);
lv_cont_set_fit(box1, true, true);

/*Add a text to the container*/
lv_obj_t * txt = lv_label_create(box1, NULL);
lv_label_set_text(txt, "Lorem ipsum dolor\n"
                      "sit amet, consectetur\n"
                      "adipiscing elit, sed do\n"
                      "eiusmod tempor incididunt\n"
                      "ut labore et dolore\n"
                      "magna aliqua.");

lv_obj_align(box1, NULL, LV_ALIGN_IN_TOP_LEFT, 10, 10);      /*Align the container*/

/*Create a style*/
static lv_style_t style;
lv_style_copy(&style, &lv_style_pretty_color);
style.body.shadow.width = 6;
style.body.padding.hor = 5;                                /*Set a great horizontal_
↳padding*/

/*Create an other container*/
lv_obj_t * box2;
box2 = lv_cont_create(lv_scr_act(), NULL);
lv_obj_set_style(box2, &style);      /*Set the new style*/
lv_cont_set_fit(box2, true, false); /*Do not enable the vertical fit */
lv_obj_set_height(box2, 55);        /*Set a fix height*/

/*Add a text to the new container*/
lv_obj_t * txt2 = lv_label_create(box2, NULL);
lv_label_set_text(txt2, "No vertical fit 1...\n"
                      "No vertical fit 2...\n"
                      "No vertical fit 3...\n"
                      "No vertical fit 4...");

/*Align the container to the bottom of the previous*/
lv_obj_align(box2, box1, LV_ALIGN_OUT_BOTTOM_MID, 30, -30);
```

MicroPython

No examples yet.

5.10.7 API

Typedefs

```
typedef uint8_t lv_layout_t
typedef uint8_t lv_fit_t
typedef uint8_t lv_cont_style_t
```

Enums

enum [anonymous]

Values:

LV_LAYOUT_OFF = 0
 LV_LAYOUT_CENTER
 LV_LAYOUT_COL_L
 LV_LAYOUT_COL_M
 LV_LAYOUT_COL_R
 LV_LAYOUT_ROW_T
 LV_LAYOUT_ROW_M
 LV_LAYOUT_ROW_B
 LV_LAYOUT_PRETTY
 LV_LAYOUT_GRID

enum [anonymous]

Values:

LV_FIT_NONE
 LV_FIT_TIGHT
 LV_FIT_FLOOD
 LV_FIT_FILL

enum [anonymous]

Values:

LV_CONT_STYLE_MAIN

Functions

lv_obj_t *lv_cont_create (lv_obj_t *par, **const** lv_obj_t *copy)

Create a container objects

Return pointer to the created container

Parameters

- par: pointer to an object, it will be the parent of the new container
- copy: pointer to a container object, if not NULL then the new object will be copied from it

void lv_cont_set_layout (lv_obj_t *cont, *lv_layout_t* layout)

Set a layout on a container

Parameters

- cont: pointer to a container object
- layout: a layout from 'lv_cont_layout_t'

void lv_cont_set_fit4 (lv_obj_t *cont, *lv_fit_t* left, *lv_fit_t* right, *lv_fit_t* top, *lv_fit_t* bottom)

Set the fit policy in all 4 directions separately. It tell how to change the container's size automatically.

Parameters

- `cont`: pointer to a container object
- `left`: left fit policy from `lv_fit_t`
- `right`: right fit policy from `lv_fit_t`
- `top`: bottom fit policy from `lv_fit_t`
- `bottom`: bottom fit policy from `lv_fit_t`

static void `lv_cont_set_fit2` (`lv_obj_t` **cont*, *lv_fit_t* hor, *lv_fit_t* ver)

Set the fit policy horizontally and vertically separately. It tell how to change the container's size automatically.

Parameters

- `cont`: pointer to a container object
- `hor`: horizontal fit policy from `lv_fit_t`
- `ver`: vertical fit policy from `lv_fit_t`

static void `lv_cont_set_fit` (`lv_obj_t` **cont*, *lv_fit_t* fit)

Set the fit policy in all 4 direction at once. It tell how to change the container's size automatically.

Parameters

- `cont`: pointer to a container object
- `fit`: fit policy from `lv_fit_t`

static void `lv_cont_set_style` (`lv_obj_t` **cont*, *lv_cont_style_t* type, **const** `lv_style_t` **style*)

Set the style of a container

Parameters

- `cont`: pointer to a container object
- `type`: which style should be set (can be only `LV_CONT_STYLE_MAIN`)
- `style`: pointer to the new style

lv_layout_t `lv_cont_get_layout` (**const** `lv_obj_t` **cont*)

Get the layout of a container

Return the layout from 'lv_cont_layout_t'

Parameters

- `cont`: pointer to container object

lv_fit_t `lv_cont_get_fit_left` (**const** `lv_obj_t` **cont*)

Get left fit mode of a container

Return an element of `lv_fit_t`

Parameters

- `cont`: pointer to a container object

lv_fit_t `lv_cont_get_fit_right` (**const** `lv_obj_t` **cont*)

Get right fit mode of a container

Return an element of `lv_fit_t`

Parameters

- `cont`: pointer to a container object

```
lv_fit_t lv_cont_get_fit_top (const lv_obj_t *cont)
```

Get top fit mode of a container

Return an element of *lv_fit_t*

Parameters

- *cont*: pointer to a container object

```
lv_fit_t lv_cont_get_fit_bottom (const lv_obj_t *cont)
```

Get bottom fit mode of a container

Return an element of *lv_fit_t*

Parameters

- *cont*: pointer to a container object

```
static const lv_style_t *lv_cont_get_style (const lv_obj_t *cont, lv_cont_style_t type)
```

Get the style of a container

Return pointer to the container's style

Parameters

- *cont*: pointer to a container object
- *type*: which style should be get (can be only LV_CONT_STYLE_MAIN)

```
struct lv_cont_ext_t
```

#include <lv_cont.h>

Public Members

uint8_t *layout*

uint8_t *fit_left*

uint8_t *fit_right*

uint8_t *fit_top*

uint8_t *fit_bottom*

5.11 Drop down list (lv_ddlist)

5.11.1 Overview

Drop Down Lists allow you to simply **select one option from more**. The Drop Down List is closed by default and show the currently selected text. If you click on it the this list opens and all the options are shown.

The **options** are passed to the Drop Down List as a **string** with `lv_ddlist_set_options(ddlist, options)`. The options should be separated by `\n`. For example: "First\nSecond\nThird".

You can **select an option manually** with `lv_ddlist_set_selected(ddlist, id)`, where *id* is the index of an option.

By default the list's **height** is adjusted automatically to show all options. The `lv_ddlist_set_fix_height(ddlist, height)` sets a fixed height for the opened list. 0 means to use auto height.

The **width** is also adjusted automatically. To prevent this apply `lv_ddlist_set_fix_width(ddlist, width)`. 0 means to use auto width.

Similarly to [Page](#) with fix height the Drop Down List supports various **scrollbar display modes**. It can be set by `lv_ddlist_set_sb_mode(ddlist, LV_SB_MODE_...)`.

The Drop Down List open/close animation time is adjusted by `lv_ddlist_set_anim_time(ddlist, anim_time)`. Zero animation time means no animation.

A **down arrow** can be added to the left side of the drop down list with `lv_ddlist_set_draw_arrow(ddlist, true)`.

To align the label horizontally use `lv_ddlist_set_align(ddlist, LV_LABEL_ALIGN_LEFT/CENTER/RIGHT)`.

You can force the Drop down list to **stay opened** when an option is selected with `lv_ddlist_set_stay_open(ddlist, true)`.

5.11.2 Styles

The `lv_ddlist_set_style(ddlist, LV_DDLIST_STYLE_..., &style)` set the styles of a Drop Down List.

- **LV_DDLIST_STYLE_BG** Style of the background. All *style.body* properties are used. It is used for the label's style from *style.text*. Default: *lv_style_pretty*
- **LV_DDLIST_STYLE_SEL** Style of the selected option. The *style.body* properties are used. The selected option will be recolored with *text.color*. Default: *lv_style_plain_color*
- **LV_DDLIST_STYLE_SB** Style of the scrollbar. The *style.body* properties are used. Default: *lv_style_plain_color*

5.11.3 Events

Besides the [Generic events](#) the following [Special events](#) are sent by the Drop down lists:

- **LV_EVENT_VALUE_CHANGED** sent when the a new option is selected

Learn more about [Events](#).

5.11.4 Keys

The following *Keys* are processed by the Buttons:

- **LV_KEY_RIGHT/DOWN** Select the next option
- **LV_KEY_LEFT/UP** Select the previous option
- **LV_KEY_ENTER** Apply the selected option (Send `LV_EVENT_VALUE_CHANGED` event and close the Drop down list)

5.11.5 Example

C



Drop down list image

code

```
static lv_res_t ddlist_action(lv_obj_t * ddlist)
{
    uint8_t id = lv_obj_get_free_num(ddlist);

    char sel_str[32];
    lv_ddlist_get_selected_str(ddlist, sel_str);
    printf("Ddlist %d new option: %s \n", id, sel_str);

    return LV_RES_OK; /*Return OK if the drop down list is not deleted*/
}

.
.
.

/*Create a drop down list*/
lv_obj_t * ddl1 = lv_ddlist_create(lv_scr_act(), NULL);
lv_ddlist_set_options(ddl1, "Apple\n"
                             "Banana\n"
                             "Orange\n"
                             "Melon\n"
                             "Grape\n"
                             "Raspberry");
lv_obj_align(ddl1, NULL, LV_ALIGN_IN_TOP_LEFT, 30, 10);
lv_obj_set_free_num(ddl1, 1); /*Set a unique ID*/
lv_ddlist_set_action(ddl1, ddlist_action); /*Set a function to call when anew option_
↪is chosen*/

/*Create a style*/
static lv_style_t style_bg;
```

(continues on next page)

(continued from previous page)

```
lv_style_copy(&style_bg, &lv_style_pretty);
style_bg.body.shadow.width = 4; /*Enable the shadow*/
style_bg.text.color = LV_COLOR_MAKE(0x10, 0x20, 0x50);

/*Copy the drop down list and set the new style_bg*/
lv_obj_t * ddl2 = lv_ddlist_create(lv_scr_act(), ddl1);
lv_obj_align(ddl2, NULL, LV_ALIGN_IN_TOP_RIGHT, -30, 10);
lv_obj_set_free_num(ddl2, 2); /*Set a unique ID*/
lv_obj_set_style(ddl2, &style_bg);
```

MicroPython

No examples yet.

5.11.6 API

Typedefs

```
typedef uint8_t lv_ddlist_style_t
```

Enums

```
enum [anonymous]
```

Values:

```
LV_DDLIST_STYLE_BG
```

```
LV_DDLIST_STYLE_SEL
```

```
LV_DDLIST_STYLE_SB
```

Functions

```
lv_obj_t * lv_ddlist_create (lv_obj_t *par, const lv_obj_t *copy)
```

Create a drop down list objects

Return pointer to the created drop down list

Parameters

- par: pointer to an object, it will be the parent of the new drop down list
- copy: pointer to a drop down list object, if not NULL then the new object will be copied from it

```
void lv_ddlist_set_options (lv_obj_t *ddl, const char *options)
```

Set the options in a drop down list from a string

Parameters

- ddl: pointer to drop down list object
- options: a string with ‘ ’ separated options. E.g. “One\nTwo\nThree”

```
void lv_ddlist_set_selected (lv_obj_t *ddl, uint16_t sel_opt)
```

Set the selected option

Parameters

- `ddlist`: pointer to drop down list object
- `sel_opt`: id of the selected option (0 ... number of option - 1);

void **lv_ddlist_set_fix_height** (lv_obj_t **ddlist*, lv_coord_t *h*)

Set a fix height for the drop down list If 0 then the opened ddlist will be auto. sized else the set height will be applied.

Parameters

- `ddlist`: pointer to a drop down list
- `h`: the height when the list is opened (0: auto size)

void **lv_ddlist_set_fix_width** (lv_obj_t **ddlist*, lv_coord_t *w*)

Set a fix width for the drop down list

Parameters

- `ddlist`: pointer to a drop down list
- `w`: the width when the list is opened (0: auto size)

void **lv_ddlist_set_draw_arrow** (lv_obj_t **ddlist*, bool *en*)

Set arrow draw in a drop down list

Parameters

- `ddlist`: pointer to drop down list object
- `en`: enable/disable a arrow draw. E.g. “true” for draw.

void **lv_ddlist_set_stay_open** (lv_obj_t **ddlist*, bool *en*)

Leave the list opened when a new value is selected

Parameters

- `ddlist`: pointer to drop down list object
- `en`: enable/disable “stay open” feature

static void **lv_ddlist_set_sb_mode** (lv_obj_t **ddlist*, lv_sb_mode_t *mode*)

Set the scroll bar mode of a drop down list

Parameters

- `ddlist`: pointer to a drop down list object
- `sb_mode`: the new mode from ‘lv_page_sb_mode_t’ enum

void **lv_ddlist_set_anim_time** (lv_obj_t **ddlist*, uint16_t *anim_time*)

Set the open/close animation time.

Parameters

- `ddlist`: pointer to a drop down list
- `anim_time`: open/close animation time [ms]

void **lv_ddlist_set_style** (lv_obj_t **ddlist*, *lv_ddlist_style_t* *type*, **const** lv_style_t **style*)

Set a style of a drop down list

Parameters

- `ddlist`: pointer to a drop down list object
- `type`: which style should be set

- `style`: pointer to a style

void **lv_ddlist_set_align** (lv_obj_t **ddlist*, lv_label_align_t *align*)
Set the alignment of the labels in a drop down list

Parameters

- `ddlist`: pointer to a drop down list object
- `align`: alignment of labels

const char ***lv_ddlist_get_options** (const lv_obj_t **ddlist*)
Get the options of a drop down list

Return the options separated by ‘ ’-s (E.g. “Option1\nOption2\nOption3”)

Parameters

- `ddlist`: pointer to drop down list object

uint16_t **lv_ddlist_get_selected** (const lv_obj_t **ddlist*)
Get the selected option

Return id of the selected option (0 ... number of option - 1);

Parameters

- `ddlist`: pointer to drop down list object

void **lv_ddlist_get_selected_str** (const lv_obj_t **ddlist*, char **buf*, uint16_t *buf_size*)
Get the current selected option as a string

Parameters

- `ddlist`: pointer to ddlist object
- `buf`: pointer to an array to store the string
- `buf_size`: size of `buf` in bytes. 0: to ignore it.

lv_coord_t **lv_ddlist_get_fix_height** (const lv_obj_t **ddlist*)
Get the fix height value.

Return the height if the ddlist is opened (0: auto size)

Parameters

- `ddlist`: pointer to a drop down list object

bool **lv_ddlist_get_draw_arrow** (lv_obj_t **ddlist*)
Get arrow draw in a drop down list

Parameters

- `ddlist`: pointer to drop down list object

bool **lv_ddlist_get_stay_open** (lv_obj_t **ddlist*)
Get whether the drop down list stay open after selecting a value or not

Parameters

- `ddlist`: pointer to drop down list object

static lv_sb_mode_t **lv_ddlist_get_sb_mode** (const lv_obj_t **ddlist*)
Get the scroll bar mode of a drop down list

Return scrollbar mode from ‘lv_page_sb_mode_t’ enum

Parameters

- `ddlist`: pointer to a drop down list object

`uint16_t lv_ddlist_get_anim_time (const lv_obj_t *ddlist)`

Get the open/close animation time.

Return open/close animation time [ms]

Parameters

- `ddlist`: pointer to a drop down list

`const lv_style_t *lv_ddlist_get_style (const lv_obj_t *ddlist, lv_ddlist_style_t type)`

Get a style of a drop down list

Return style pointer to a style

Parameters

- `ddlist`: pointer to a drop down list object
- `type`: which style should be get

`lv_label_align_t lv_ddlist_get_align (const lv_obj_t *ddlist)`

Get the alignment of the labels in a drop down list

Return alignment of labels

Parameters

- `ddlist`: pointer to a drop down list object

`void lv_ddlist_open (lv_obj_t *ddlist, lv_anim_enable_t anim)`

Open the drop down list with or without animation Open the drop down list with or without animation

Parameters

- `ddlist`: pointer to drop down list object
- `anim_en`: LV_ANIM_ON: use animation; LV_ANOM_OFF: not use animations

Parameters

- `ddlist`: pointer to drop down list object
- `anim_en`: LV_ANIM_EN: use animation; LV_ANIM_OFF: not use animations

`void lv_ddlist_close (lv_obj_t *ddlist, lv_anim_enable_t anim)`

Close (Collapse) the drop down list Close (Collapse) the drop down list

Parameters

- `ddlist`: pointer to drop down list object
- `anim_en`: LV_ANIM_ON: use animation; LV_ANOM_OFF: not use animations

Parameters

- `ddlist`: pointer to drop down list object
- `anim_en`: LV_ANIM_ON: use animation; LV_ANIM_OFF: not use animations

`struct lv_ddlist_ext_t`

`#include <lv_ddlist.h>`

Public Members

```
lv_page_ext_t page
lv_obj_t *label
const lv_style_t *sel_style
uint16_t option_cnt
uint16_t sel_opt_id
uint16_t sel_opt_id_ori
uint16_t anim_time
uint8_t opened
uint8_t force_sel
uint8_t draw_arrow
uint8_t stay_open
lv_coord_t fix_height
```

5.12 Gauge (lv_gauge)

5.12.1 Overview

The gauge is a meter with **scale labels** and **needles**. You can use the `lv_gauge_set_scale(gauge, angle, line_num, label_cnt)` function to adjust the scale angle and the number of the scale lines and labels. The default settings are: 220 degrees, 6 scale labels and 21 lines.

The gauge can show **more than one needles**. Use the `lv_gauge_set_needle_count(gauge, needle_num, color_array)` function to set the number of needles and an array with colors for each needle. The array must be static or global variable because only its pointer is stored.

You can use `lv_gauge_set_value(gauge, needle_id, value)` to **set the value of a needle**.

To set a **critical value** use `lv_gauge_set_critical_value(gauge, value)`. The scale color will be changed to `line.color` after this value. (default: 80)

The **range** of the gauge can be specified by `lv_gauge_set_range(gauge, min, max)`. The default range is 0..100.

5.12.2 Styles

The gauge uses one style which can be set by `lv_gauge_set_style(gauge, LV_GAUGE_STYLE_MAIN, &style)`. The gauge's properties are derived from the following style attributes:

- **body.main_color** line's color at the beginning of the scale
- **body.grad_color** line's color at the end of the scale (gradient with main color)
- **body.padding.hor** line length
- **body.padding.inner** label distance from the scale lines
- **body.radius** radius of needle origin circle
- **line.width** line width

- **line.color** line's color after the critical value
- **text.font/color/letter_space** label attributes

5.12.3 Events

Only the [Generic events](#) are sent by the object type.

Learn more about [Events](#).

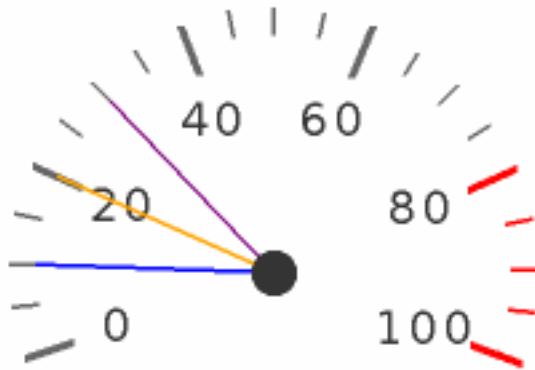
5.12.4 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.12.5 Example

C



Gauge image

code

```
/*Create a style*/
static lv_style_t style;
lv_style_copy(&style, &lv_style_pretty_color);
style.body.main_color = LV_COLOR_HEX3(0x666); /*Line color at the beginning*/
style.body.grad_color = LV_COLOR_HEX3(0x666); /*Line color at the end*/
style.body.padding.hor = 10; /*Scale line length*/
style.body.padding.inner = 8; /*Scale label padding*/
style.body.border.color = LV_COLOR_HEX3(0x333); /*Needle middle circle color*/
style.line.width = 3;
style.text.color = LV_COLOR_HEX3(0x333);
```

(continues on next page)

(continued from previous page)

```
style.line.color = LV_COLOR_RED;           /*Line color after the critical_
↪value*/

/*Describe the color for the needles*/
static lv_color_t needle_colors[] = {LV_COLOR_BLUE, LV_COLOR_ORANGE, LV_COLOR_PURPLE};

/*Create a gauge*/
lv_obj_t * gauge1 = lv_gauge_create(lv_scr_act(), NULL);
lv_gauge_set_style(gauge1, &style);
lv_gauge_set_needle_count(gauge1, 3, needle_colors);
lv_obj_align(gauge1, NULL, LV_ALIGN_CENTER, 0, 20);

/*Set the values*/
lv_gauge_set_value(gauge1, 0, 10);
lv_gauge_set_value(gauge1, 1, 20);
lv_gauge_set_value(gauge1, 2, 30);
```

MicroPython

No examples yet.

5.12.6 API

Typedefs

```
typedef uint8_t lv_gauge_style_t
```

Enums

```
enum [anonymous]
```

Values:

```
LV_GAUGE_STYLE_MAIN
```

Functions

```
lv_obj_t *lv_gauge_create(lv_obj_t *par, const lv_obj_t *copy)
```

Create a gauge objects

Return pointer to the created gauge

Parameters

- `par`: pointer to an object, it will be the parent of the new gauge
- `copy`: pointer to a gauge object, if not NULL then the new object will be copied from it

```
void lv_gauge_set_needle_count(lv_obj_t *gauge, uint8_t needle_cnt, const lv_color_t colors[])
```

Set the number of needles

Parameters

- `gauge`: pointer to gauge object

- `needle_cnt`: new count of needles
- `colors`: an array of colors for needles (with ‘num’ elements)

void **lv_gauge_set_value** (lv_obj_t **gauge*, uint8_t *needle_id*, int16_t *value*)
Set the value of a needle

Parameters

- `gauge`: pointer to a gauge
- `needle_id`: the id of the needle
- `value`: the new value

static void **lv_gauge_set_range** (lv_obj_t **gauge*, int16_t *min*, int16_t *max*)
Set minimum and the maximum values of a gauge

Parameters

- `gauge`: pointer to the gauge object
- `min`: minimum value
- `max`: maximum value

static void **lv_gauge_set_critical_value** (lv_obj_t **gauge*, int16_t *value*)
Set a critical value on the scale. After this value ‘line.color’ scale lines will be drawn

Parameters

- `gauge`: pointer to a gauge object
- `value`: the critical value

void **lv_gauge_set_scale** (lv_obj_t **gauge*, uint16_t *angle*, uint8_t *line_cnt*, uint8_t *label_cnt*)
Set the scale settings of a gauge

Parameters

- `gauge`: pointer to a gauge object
- `angle`: angle of the scale (0..360)
- `line_cnt`: count of scale lines. The get a given “subdivision” lines between label, $line_cnt = (sub_div + 1) * (label_cnt - 1) + 1$
- `label_cnt`: count of scale labels.

static void **lv_gauge_set_style** (lv_obj_t **gauge*, lv_gauge_style_t *type*, lv_style_t **style*)
Set the styles of a gauge

Parameters

- `gauge`: pointer to a gauge object
- `type`: which style should be set (can be only LV_GAUGE_STYLE_MAIN)
- `style`: set the style of the gauge

int16_t **lv_gauge_get_value** (const lv_obj_t **gauge*, uint8_t *needle*)
Get the value of a needle

Return the value of the needle [min,max]

Parameters

- `gauge`: pointer to gauge object

- `needle`: the id of the needle

`uint8_t lv_gauge_get_needle_count (const lv_obj_t *gauge)`

Get the count of needles on a gauge

Return count of needles

Parameters

- `gauge`: pointer to gauge

`static int16_t lv_gauge_get_min_value (const lv_obj_t *lmeter)`

Get the minimum value of a gauge

Return the minimum value of the gauge

Parameters

- `gauge`: pointer to a gauge object

`static int16_t lv_gauge_get_max_value (const lv_obj_t *lmeter)`

Get the maximum value of a gauge

Return the maximum value of the gauge

Parameters

- `gauge`: pointer to a gauge object

`static int16_t lv_gauge_get_critical_value (const lv_obj_t *gauge)`

Get a critical value on the scale.

Return the critical value

Parameters

- `gauge`: pointer to a gauge object

`uint8_t lv_gauge_get_label_count (const lv_obj_t *gauge)`

Set the number of labels (and the thicker lines too)

Return count of labels

Parameters

- `gauge`: pointer to a gauge object

`static uint8_t lv_gauge_get_line_count (const lv_obj_t *gauge)`

Get the scale number of a gauge

Return number of the scale units

Parameters

- `gauge`: pointer to a gauge object

`static uint16_t lv_gauge_get_scale_angle (const lv_obj_t *gauge)`

Get the scale angle of a gauge

Return angle of the scale

Parameters

- `gauge`: pointer to a gauge object

`static const lv_style_t *lv_gauge_get_style (const lv_obj_t *gauge, lv_gauge_style_t type)`

Get the style of a gauge

Return pointer to the gauge's style

Parameters

- gauge: pointer to a gauge object
- type: which style should be get (can be only LV_GAUGE_STYLE_MAIN)

```
struct lv_gauge_ext_t
#include <lv_gauge.h>
```

Public Members

```
lv_lmeter_ext_t lmeter
int16_t *values
const lv_color_t *needle_colors
uint8_t needle_count
uint8_t label_count
```

5.13 Image (lv_img)

5.13.1 Overview

The Images are the basic object to **display images**. To provide maximum flexibility the **source of the image** can be:

- a variable in the code (a C array with the pixels)
- a file stored externally (like on an SD card)
- a text with *Symbols*

To set the source of an image use `lv_img_set_src(img, src)`

To generate a pixel array **from a PNG, JPG or BMP** image use the [Online image converter tool](#) and set the converted image with its pointer: `lv_img_set_src(img1, &converted_img_var)`; To make the variable visible in the C file you need to declare it with `LV_IMG_DECLARE(converted_img_var)`

To use **external files** you also need to convert the image files using the online converter tool but now you should select the binary Output format. You also need to use LittlevGL's file system module and register a driver with some functions for the basic file operation. Got to the [File system](#) to learn more. To set an image source from a file use `lv_img_set_src(img, "S:folder1/my_img.bin")`

You can set a **symbol** similarly to *Labels*. In this case, the image will be rendered as text according to the *font* specified in the style. It enables to use light weighted mono-color “letters” instead of real images. You can set symbol like `lv_img_set_src(img1, LV_SYMBOL_OK)`

Images and labels are sometimes for the same thing. E.g. to describe what a button does. Therefore Images and Labels are somewhat interchangeable. To handle this images can even display **texts** by using `LV_SYMBOL_DUMMY` as prefix of the text. For example `lv_img_set_src(img, LV_SYMBOL_DUMMY "Some text")`

The internal (variable) and external images support 2 **transparency handling** methods:

- **Chrome keying** pixels with `LV_COLOR_TRANSP` (lv_conf.h) color will be transparent
- **Alpha byte** An alpha byte is added to every pixel

Besides *True color* color format the following format are also supported:

- **Indexed** image has a palette

- **Alpha indexed** only alpha values are stored

These options can be selected in the font converter. To learn more about the color formats read the [Images](#) section.

The images can be **re-colored in run-time** to any color according to the brightness of the pixels. It is very useful to show different states (selected, inactive, pressed etc) of an image without storing more versions of the same image. This feature can be enabled in the style by setting `img.intense` between `LV_OPA_TRANSP` (no recolor, value: 0) and `LV_OPA_COVER` (full recolor, value: 255). The default value is `LV_OPA_TRANSP` so this feature is disabled.

It is possible to **automatically set the size** of the image object to the image source's width and height if enabled by the `lv_img_set_auto_size(image, true)` function. If *auto size* is enabled then when a new file is set the object size is automatically changed. Later you can modify the size manually. The *auto size* is enabled by default if the image is not a screen

If the object size is greater then the image size in any directions then the image will be repeated like a mosaic. It's a very useful a feature to create a large image from only a very narrow source. For example you can have a *300 x 1* image with a special gradient and set it as a wallpaper using the mosaic feature.

The images' default style is *NULL* so they **inherit the parent's style**.

5.13.2 Styles

The images uses one style which can be set by `lv_img_set_style(lmeter, LV_IMG_STYLE_MAIN, &style)`. All the `style.image` propertis are used:

- **image.inense** intensity of recoloring (0..255 or *LV_OPA_...*)
- **image.color** color for recoloring or color of the alpha indexed images
- **image.opa** overall opacitiy of image

When the Image obejct idplays a text then `style.text` properties are used. See [Label](#) for more information.

5.13.3 Events

Only the [Generic events](#) are sent by the object type.

Learn more about [Events](#).

5.13.4 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.13.5 Example

C

Re-color the images in run time



Use symbols from fonts as images



Example of Image in LittlevGL

Graphics Library

code

```
/*Create the first image without re-color*/
lv_obj_t * img1 = lv_img_create(lv_scr_act(), NULL);
lv_img_set_src(img1, &img_cw);
lv_obj_align(img1, NULL, LV_ALIGN_IN_TOP_LEFT, 20, 40);

/*Create style to re-color with light blue*/
static lv_style_t style_img2;
lv_style_copy(&style_img2, &lv_style_plain);
style_img2.image.color = LV_COLOR_HEX(0x003b75);
style_img2.image.intense = LV_OPA_50;

/*Create an image with the light blue style*/
lv_obj_t * img2 = lv_img_create(lv_scr_act(), img1);
lv_obj_set_style(img2, &style_img2);
lv_obj_align(img2, NULL, LV_ALIGN_IN_TOP_MID, 0, 40);

/*Create style to re-color with dark blue*/
static lv_style_t style_img3;
lv_style_copy(&style_img3, &lv_style_plain);
style_img3.image.color = LV_COLOR_HEX(0x003b75);
style_img3.image.intense = LV_OPA_90;

/*Create an image with the dark blue style*/
lv_obj_t * img3 = lv_img_create(lv_scr_act(), img2);
lv_obj_set_style(img3, &style_img3);
lv_obj_align(img3, NULL, LV_ALIGN_IN_TOP_RIGHT, -20, 40);

/*****
```

(continues on next page)

(continued from previous page)

```

* Create an image with symbols
*****/

/*Create a string from symbols*/
char buf[32];
sprintf(buf, "%s%s%s%s%s%s",
        SYMBOL_DRIVE, SYMBOL_FILE, SYMBOL_DIRECTORY, SYMBOL_SETTINGS,
        SYMBOL_POWER, SYMBOL_GPS, SYMBOL_BLUETOOTH);

/*Create style with a symbol font*/
static lv_style_t style_sym;
lv_style_copy(&style_sym, &lv_style_plain);
// The built-in fonts are extended with symbols
style_sym.text.font = &lv_font_dejavu_60;
style_sym.text.letter_space = 10;

/*Create an image and use the string as source*/
lv_obj_t * img_sym = lv_img_create(lv_scr_act(), NULL);
lv_img_set_src(img_sym, buf);
lv_img_set_style(img_sym, &style_sym);
lv_obj_align(img_sym, NULL, LV_ALIGN_IN_BOTTOM_MID, 0, -30);

/*Create description labels*/
lv_obj_t * label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label, "Re-color the images in run time");
lv_obj_align(label, NULL, LV_ALIGN_IN_TOP_MID, 0, 15);

label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label, "Use symbols from fonts as images");
lv_obj_align(label, NULL, LV_ALIGN_IN_BOTTOM_MID, 0, -80);

```

MicroPython

No examples yet.

5.13.6 API

Defines

LV_IMG_DECLARE (var_name)

Typedefs

typedef uint8_t **lv_img_style_t**

Enums

enum [anonymous]

Values:

LV_IMG_STYLE_MAIN

Functions

`lv_obj_t *lv_img_create (lv_obj_t *par, const lv_obj_t *copy)`

Create an image objects

Return pointer to the created image

Parameters

- `par`: pointer to an object, it will be the parent of the new button
- `copy`: pointer to a image object, if not NULL then the new object will be copied from it

`void lv_img_set_src (lv_obj_t *img, const void *src_img)`

Set the pixel map to display by the image

Parameters

- `img`: pointer to an image object
- `data`: the image data

`void lv_img_set_auto_size (lv_obj_t *img, bool autosize_en)`

Enable the auto size feature. If enabled the object size will be same as the picture size.

Parameters

- `img`: pointer to an image
- `en`: true: auto size enable, false: auto size disable

`void lv_img_set_offset_x (lv_obj_t *img, lv_coord_t x)`

Set an offset for the source of an image. so the image will be displayed from the new origin.

Parameters

- `img`: pointer to an image
- `x`: the new offset along x axis.

`void lv_img_set_offset_y (lv_obj_t *img, lv_coord_t y)`

Set an offset for the source of an image. so the image will be displayed from the new origin.

Parameters

- `img`: pointer to an image
- `y`: the new offset along y axis.

`static void lv_img_set_style (lv_obj_t *img, lv_img_style_t type, const lv_style_t *style)`

Set the style of an image

Parameters

- `img`: pointer to an image object
- `type`: which style should be set (can be only LV_IMG_STYLE_MAIN)
- `style`: pointer to a style

`const void *lv_img_get_src (lv_obj_t *img)`

Get the source of the image

Return the image source (symbol, file name or C array)

Parameters

- `img`: pointer to an image object

const char ***lv_img_get_file_name** (**const** lv_obj_t *img)

Get the name of the file set for an image

Return file name

Parameters

- img: pointer to an image

bool **lv_img_get_auto_size** (**const** lv_obj_t *img)

Get the auto size enable attribute

Return true: auto size is enabled, false: auto size is disabled

Parameters

- img: pointer to an image

lv_coord_t **lv_img_get_offset_x** (lv_obj_t *img)

Get the offset.x attribute of the img object.

Return offset.x value.

Parameters

- img: pointer to an image

lv_coord_t **lv_img_get_offset_y** (lv_obj_t *img)

Get the offset.y attribute of the img object.

Return offset.y value.

Parameters

- img: pointer to an image

static const lv_style_t ***lv_img_get_style** (**const** lv_obj_t *img, *lv_img_style_t* type)

Get the style of an image object

Return pointer to the image's style

Parameters

- img: pointer to an image object
- type: which style should be get (can be only LV_IMG_STYLE_MAIN)

struct **lv_img_ext_t**

#include <lv_img.h>

Public Members

const void ***src**

lv_point_t **offset**

lv_coord_t **w**

lv_coord_t **h**

uint8_t **src_type**

uint8_t **auto_size**

uint8_t **cf**

5.14 Image button (lv_imgbtn)

5.14.1 Overview

The Image button is very similar to the simple Button object. The only difference is it displays user-defined images in each state instead of drawing a button. Before reading this please read the [Button](#) section too.

To set the image in a state the `lv_imgbtn_set_src(imgbtn, LV_BTN_STATE_..., &img_src)` The image sources works the same as described in the [Image object](#).

If `LV_IMGBTN_TILED` is enabled in `lv_conf.h` three source can be set for state:

- left
- center
- right

The *center* image will repeated to fill the width of object. Therefore with `LV_IMGBTN_TILED` you can set the width of the Image button while without it the width will be always the same as the image source's width.

The **states** also work like with Button object. It can be set with `lv_imgbtn_set_state(imgbtn, LV_BTN_STATE_...)`.

The **toggle** feature can be enabled with `lv_imgbtn_set_toggle(imgbtn, true)`

5.14.2 Style usage

Similarly to normal Buttons, Image buttons also have 5 independent styles for the 5 state. You can set them via: `lv_imgbtn_set_style(btn, LV_IMGBTN_STYLE_..., &style)`. The styles use the `style.image` properties.

- `LV_IMGBTN_STYLE_REL` style of the released state. Default: `lv_style_btn_rel`
- `LV_IMGBTN_STYLE_PR` style of the pressed state. Default: `lv_style_btn_pr`
- `LV_IMGBTN_STYLE_TGL_REL` style of the toggled released state. Default: `lv_style_btn_tgl_rel`
- `LV_IMGBTN_STYLE_TGL_PR` style of the toggled pressed state. Default: `lv_style_btn_tgl_pr`
- `LV_IMGBTN_STYLE_INA` style of the inactive state. Default: `lv_style_btn_ina`

When labels are created on a button, it's a good practice to set the image button's `style.text` properties too. Because labels have `style = NULL` by default they inherit the parent's (image button) style. Hence you don't need to create a new style for the label.

5.14.3 Events

Besides the [Generic events](#) the following [Special events](#) are sent by the buttons:

- `LV_EVENT_VALUE_CHANGED` sent when the button is toggled.

Note that the generic input device related events (like `LV_EVENT_PRESSED`) are sent in inactive state too. You need to check the state with `lv_btn_get_state(btn)` to ignore the events from inactive buttons.

Learn more about [Events](#).

5.14.4 Keys

The following *Keys* are processed by the Buttons:

- **LV_KEY_RIGHT/UP** Go to toggled state if toggling is enabled
- **LV_KEY_LEFT/DOWN** Go to non-toggled state if toggling is enabled

Note that, as usual, the state of LV_KEY_ENTER is translated to LV_EVENT_PRESSED/PRESSING/RELEASED etc.

Learn more about [Keys](#).

5.14.5 Example

C



Image button image

code

```
/*Create style to make the button darker when pressed*/
lv_style_t style_pr;
lv_style_copy(&style_pr, &lv_style_plain);
style_pr.image.color = LV_COLOR_BLACK;
style_pr.image.intense = LV_OPA_50;
style_pr.text.color = LV_COLOR_HEX3(0xaaa);

LV_IMG_DECLARE(imgbtn_green);
LV_IMG_DECLARE(imgbtn_blue);

/*Create an Image button*/
lv_obj_t * imgbtn1 = lv_imgbtn_create(lv_scr_act(), NULL);
lv_imgbtn_set_src(imgbtn1, LV_BTN_STATE_REL, &imgbtn_green);
lv_imgbtn_set_src(imgbtn1, LV_BTN_STATE_PR, &imgbtn_green);
lv_imgbtn_set_src(imgbtn1, LV_BTN_STATE_TGL_REL, &imgbtn_blue);
lv_imgbtn_set_src(imgbtn1, LV_BTN_STATE_TGL_PR, &imgbtn_blue);
```

(continues on next page)

(continued from previous page)

```
lv_imgbtn_set_style(imgbtn1, LV_BTN_STATE_PR, &style_pr);           /*Use the darker_
↪style in the pressed state*/
lv_imgbtn_set_style(imgbtn1, LV_BTN_STATE_TGL_PR, &style_pr);
lv_imgbtn_set_toggle(imgbtn1, true);
lv_obj_align(imgbtn1, NULL, LV_ALIGN_CENTER, 0, -40);

/*Create a label on the Image button*/
lv_obj_t * label = lv_label_create(imgbtn1, NULL);
lv_label_set_text(label, "Button");

/*Copy the first image button and set Toggled state*/
lv_obj_t * imgbtn2 = lv_imgbtn_create(lv_scr_act(), imgbtn1);
lv_btn_set_state(imgbtn2, LV_BTN_STATE_TGL_REL);
lv_obj_align(imgbtn2, imgbtn1, LV_ALIGN_OUT_BOTTOM_MID, 0, 20);

/*Create a label on the Image button*/
label = lv_label_create(imgbtn2, NULL);
lv_label_set_text(label, "Button");
```

MicroPython

No examples yet.

5.14.6 API

Typedefs

```
typedef uint8_t lv_imgbtn_style_t
```

Enums

```
enum [anonymous]
```

Values:

```
LV_IMGBTN_STYLE_REL
LV_IMGBTN_STYLE_PR
LV_IMGBTN_STYLE_TGL_REL
LV_IMGBTN_STYLE_TGL_PR
LV_IMGBTN_STYLE_INA
```

Functions

```
lv_obj_t *lv_imgbtn_create(lv_obj_t *par, const lv_obj_t *copy)
```

Create a image button objects Create a image button object

Return pointer to the created image button

Return pointer to the created image button

Parameters

- `par`: pointer to an object, it will be the parent of the new image button
- `copy`: pointer to a image button object, if not NULL then the new object will be copied from it

Parameters

- `par`: pointer to an object, it will be the parent of the new image button
- `copy`: pointer to a image button object, if not NULL then the new object will be copied from it

void **lv_imgbtn_set_src** (lv_obj_t **imgbtn*, *lv_btn_state_t state*, **const** void **src*)
Set images for a state of the image button

Parameters

- `imgbtn`: pointer to an image button object
- `state`: for which state set the new image (from `lv_btn_state_t`) ‘
- `src`: pointer to an image source (a C array or path to a file)

static void **lv_imgbtn_set_toggle** (lv_obj_t **imgbtn*, bool *tgl*)
Enable the toggled states. On release the button will change from/to toggled state.

Parameters

- `imgbtn`: pointer to an image button object
- `tgl`: true: enable toggled states, false: disable

static void **lv_imgbtn_set_state** (lv_obj_t **imgbtn*, *lv_btn_state_t state*)
Set the state of the image button

Parameters

- `imgbtn`: pointer to an image button object
- `state`: the new state of the button (from `lv_btn_state_t` enum)

static void **lv_imgbtn_toggle** (lv_obj_t **imgbtn*)
Toggle the state of the image button (ON->OFF, OFF->ON)

Parameters

- `imgbtn`: pointer to a image button object

void **lv_imgbtn_set_style** (lv_obj_t **imgbtn*, *lv_imgbtn_style_t type*, **const** lv_style_t **style*)
Set a style of a image button.

Parameters

- `imgbtn`: pointer to image button object
- `type`: which style should be set
- `style`: pointer to a style

const void ***lv_imgbtn_get_src** (lv_obj_t **imgbtn*, *lv_btn_state_t state*)
Get the images in a given state

Return pointer to an image source (a C array or path to a file)

Parameters

- `imgbtn`: pointer to an image button object
- `state`: the state where to get the image (from `lv_btn_state_t`) ‘

```
static lv_btn_state_t lv_imgbtn_get_state (const lv_obj_t *imgbtn)
```

Get the current state of the image button

Return the state of the button (from lv_btn_state_t enum)

Parameters

- imgbtn: pointer to a image button object

```
static bool lv_imgbtn_get_toggle (const lv_obj_t *imgbtn)
```

Get the toggle enable attribute of the image button

Return ture: toggle enabled, false: disabled

Parameters

- imgbtn: pointer to a image button object

```
const lv_style_t *lv_imgbtn_get_style (const lv_obj_t *imgbtn, lv_imgbtn_style_t type)
```

Get style of a image button.

Return style pointer to the style

Parameters

- imgbtn: pointer to image button object
- type: which style should be get

```
struct lv_imgbtn_ext_t
```

```
#include <lv_imgbtn.h>
```

Public Members

```
lv_btn_ext_t btn
```

```
const void *img_src[LV_BTN_STATE_NUM]
```

```
lv_img_cf_t act_cf
```

5.15 Keyboard (lv_kb)

5.15.1 Overview

The Keyboard object is a special *Button matrix* with predefined key maps and other features to realize a virtual keyboard to write text.

The Keyboards have two **modes**:

- **LV_KB_MODE_TEXT** display letters, number and special characters
- **LV_KB_MODE_NUM** display numbers, +/- sign and decimal dot

To set the mode use `lv_kb_set_mode(kb, mode)`. The default is `LV_KB_MODE_TEXT`

You can assign a *Text area* to the Keyboard to automatically put the clicked characters there. To assign the Text area use `lv_kb_set_ta(kb, ta)`.

The assigned Text area's **cursor** can be **managed** by the keyboard: when the keyboard is assigned the previous Text area's cursor will be hidden and the new's will be shown. When the keyboard is closed by the *Ok* or *Close* buttons the cursor also will be hidden. The cursor manager feature is enabled by `lv_kb_set_cursor_manage(kb, true)`. The default is not manage.

You can specify a **new map** (layout) for the keyboard with `lv_kb_set_map(kb, map)`. and `lv_kb_set_ctrl_map(kb, ctrl_map)`. Learn more about in the [Button matrix](#) object. Keep in mind using following keywords will have the same effect as with the original map:

- `LV_SYMBOL_OK` Apply
- `SYMBOL_CLOSE` Close
- `LV_SYMBOL_LEFT` Move the cursor left
- `LV_SYMBOL_RIGHT` Move the cursor right
- “`ABC`” load the uppersace map
- “`abc`” load the lower case map
- “`Enter`” new line
- “`Bkps`” Delete on the left

The keyboard has a **default event handler** callback called `lv_kb_def_event_cb`. It handles the button pressing, map changing, the assigned Text area, etc. You can completely replace it with your custom event handler but you can call `lv_kb_def_event_cb` at the beginning of your event handler to handle the same things as before.

5.15.2 Styles

The Keyboards work with 6 styles: a background and 5 button styles for each states. You can set the styles with `lv_kb_set_style(btn, LV_KB_STYLE_..., &style)`. The background and the buttons use the `style.body` properties. The labels use the `style.text` properties of the buttons’ styles.

- **LV_KB_STYLE_BG** Background style. Uses all `style.body` properties including padding Default: `lv_style_pretty`
- **LV_KB_STYLE_BTN_REL** style of the released buttons. Default: `lv_style_btn_rel`
- **LV_KB_STYLE_BTN_PR** style of the pressed buttons. Default: `lv_style_btn_pr`
- **LV_KB_STYLE_BTN_TGL_REL** style of the toggled released buttons. Default: `lv_style_btn_tgl_rel`
- **LV_KB_STYLE_BTN_TGL_PR** style of the toggled pressed buttons. Default: `lv_style_btn_tgl_pr`
- **LV_KB_STYLE_BTN_INA** style of the inactive buttons. Default: `lv_style_btn_ina`

5.15.3 Events

Besided the [Generic events](#) the following [Special events](#) are sent by the keyboards:

- **LV_EVENT_VALUE_CHANGED** sent when the button is pressed/released or repeated after long press. The event data is set to ID of the pressed/released button.
- **LV_EVENT_APPLY** the *Ok* button is clicked
- **LV_EVENT_CANCEL** the *Close* button is clicked

Learn more about [Events](#).

5.15.4 Keys

The following *Keys* are processed by the Buttons:

- **LV_KEY_RIGHT/UP/LEFT/RIGHT** To navigate among the buttons and elect one
- **LV_KEY_ENTER** To press/release the selected button

Learn more about *Keys*.

5.15.5 Examples

C



Keyboard image

code

```
/*Create styles for the keyboard*/
static lv_style_t rel_style, pr_style;

lv_style_copy(&rel_style, &lv_style_btn_rel);
rel_style.body.radius = 0;

lv_style_copy(&pr_style, &lv_style_btn_pr);
pr_style.body.radius = 0;

/*Create a keyboard and apply the styles*/
lv_obj_t *kb = lv_kb_create(lv_scr_act(), NULL);
lv_kb_set_cursor_manage(kb, true);
lv_kb_set_style(kb, LV_KB_STYLE_BG, &lv_style_transp_tight);
lv_kb_set_style(kb, LV_KB_STYLE_BTN_REL, &rel_style);
lv_kb_set_style(kb, LV_KB_STYLE_BTN_PR, &pr_style);

/*Create a text area. The keyboard will write here*/
lv_obj_t *ta = lv_ta_create(lv_scr_act(), NULL);
```

(continues on next page)

(continued from previous page)

```
lv_obj_align(ta, NULL, LV_ALIGN_IN_TOP_MID, 0, 10);
lv_ta_set_text(ta, "");

/*Assign the text area to the keyboard*/
lv_kb_set_ta(kb, ta);
```

MicroPython

No examples yet.

5.15.6 API

Typedefs

```
typedef uint8_t lv_kb_mode_t
typedef uint8_t lv_kb_style_t
```

Enums

```
enum [anonymous]
    Values:
        LV_KB_MODE_TEXT
        LV_KB_MODE_NUM

enum [anonymous]
    Values:
        LV_KB_STYLE_BG
        LV_KB_STYLE_BTN_REL
        LV_KB_STYLE_BTN_PR
        LV_KB_STYLE_BTN_TGL_REL
        LV_KB_STYLE_BTN_TGL_PR
        LV_KB_STYLE_BTN_INA
```

Functions

`lv_obj_t *lv_kb_create (lv_obj_t *par, const lv_obj_t *copy)`
 Create a keyboard objects

Return pointer to the created keyboard

Parameters

- `par`: pointer to an object, it will be the parent of the new keyboard
- `copy`: pointer to a keyboard object, if not NULL then the new object will be copied from it

`void lv_kb_set_ta (lv_obj_t *kb, lv_obj_t *ta)`
 Assign a Text Area to the Keyboard. The pressed characters will be put there.

Parameters

- kb: pointer to a Keyboard object
- ta: pointer to a Text Area object to write there

void **lv_kb_set_mode** (lv_obj_t *kb, lv_kb_mode_t mode)

Set a new a mode (text or number map)

Parameters

- kb: pointer to a Keyboard object
- mode: the mode from 'lv_kb_mode_t'

void **lv_kb_set_cursor_manage** (lv_obj_t *kb, bool en)

Automatically hide or show the cursor of the current Text Area Automatically hide or show the cursor of Text Area

Parameters

- kb: pointer to a Keyboard object
- en: true: show cursor on the current text area, false: hide cursor

Parameters

- kb: pointer to a Keyboard object
- en: true: show cursor on the current text area, false: hide cursor

static void **lv_kb_set_map** (lv_obj_t *kb, const char *map[])

Set a new map for the keyboard

Parameters

- kb: pointer to a Keyboard object
- map: pointer to a string array to describe the map. See 'lv_btnm_set_map()' for more info.

static void **lv_kb_set_ctrl_map** (lv_obj_t *kb, const lv_btnm_ctrl_t ctrl_map[])

Set the button control map (hidden, disabled etc.) for the keyboard. The control map array will be copied and so may be deallocated after this function returns.

Parameters

- kb: pointer to a keyboard object
- ctrl_map: pointer to an array of lv_btn_ctrl_t control bytes. See: lv_btnm_set_ctrl_map for more details.

void **lv_kb_set_style** (lv_obj_t *kb, lv_kb_style_t type, const lv_style_t *style)

Set a style of a keyboard

Parameters

- kb: pointer to a keyboard object
- type: which style should be set
- style: pointer to a style

lv_obj_t ***lv_kb_get_ta** (const lv_obj_t *kb)

Assign a Text Area to the Keyboard. The pressed characters will be put there.

Return pointer to the assigned Text Area object

Parameters

- kb: pointer to a Keyboard object

lv_kb_mode_t **lv_kb_get_mode** (**const** lv_obj_t *kb)

Set a new a mode (text or number map)

Return the current mode from 'lv_kb_mode_t'

Parameters

- kb: pointer to a Keyboard object

bool **lv_kb_get_cursor_manage** (**const** lv_obj_t *kb)

Get the current cursor manage mode.

Return true: show cursor on the current text area, false: hide cursor

Parameters

- kb: pointer to a Keyboard object

static const char ****lv_kb_get_map_array** (**const** lv_obj_t *kb)

Get the current map of a keyboard

Return the current map

Parameters

- kb: pointer to a keyboard object

const lv_style_t ***lv_kb_get_style** (**const** lv_obj_t *kb, *lv_kb_style_t* type)

Get a style of a keyboard

Return style pointer to a style

Parameters

- kb: pointer to a keyboard object
- type: which style should be get

void **lv_kb_def_event_cb** (lv_obj_t *kb, lv_event_t event)

Default keyboard event to add characters to the Text area and change the map. If a custom event_cb is added to the keyboard this function be called from it to handle the button clicks

Parameters

- kb: pointer to a keyboard
- event: the triggering event

struct **lv_kb_ext_t**

#include <lv_kb.h>

Public Members

lv_btnm_ext_t **btnm**

lv_obj_t ***ta**

lv_kb_mode_t **mode**

uint8_t **cursor_mng**

5.16 Label (lv_label)

5.16.1 Overview

The Labels are the basic objects to **display text**. There is no limitation in the text size because it's stored dynamically. You can modify the text in runtime at any time with `lv_label_set_text()`.

You can use `\n` to make line break. For example: `"line1\nline2\n\nline4"`

The size of the label object can be automatically expanded to the text size or the text can be manipulated according to several **long mode policies**:

- `LV_LABEL_LONG_EXPAND`: Expand the object size to the text size (Default)
- `LV_LABEL_LONG_BREAK`: Keep the object width, break (wrap) the too long lines and expand the object height
- `LV_LABEL_LONG_DOTS`: Keep the object size, break the text and write dots in the last line
- `LV_LABEL_LONG_SCROLL`: Expand the object size and scroll the text on the parent (move the label object)
- `LV_LABEL_LONG_ROLL`: Keep the size and roll just the text (not the object)

You can specify the long mode with: `lv_label_set_long_mode(label, long_mode)`

It's important to note that if you change the `LONG_MODE` the size of the label object is already expanded to the text's size. So you need to set the label's size with `lv_obj_set_size()` or `lv_obj_set_width()` after changing long mode.

Labels are able to show text from a **static array**. Use: `lv_label_set_static_text(label, char_array)`. In this case, the text is not stored in the dynamic memory but the given array is used instead. Keep in my the array can't be a local variable which destroys when the function exits.

You can also use a **raw character array** as label text. The array doesn't have to be `\0` terminated. In this case, the text will be saved to the dynamic memory. To set a raw character array use the `lv_label_set_array_text(label, char_array)` function.

The label's **text can be aligned** to the left, right or middle with `lv_label_set_align(label, LV_LABEL_ALIGN_LEFT/RIGHT/CENTER)`

You can enable to **draw a background** for the label with `lv_label_set_body_draw(label, draw)`

In the text, you can use commands to **re-color parts of the text**. For example: `"Write a #ff0000 red# word"`. This feature can be enabled individually for each label by `lv_label_set_recolor()` function.

The labels can display symbols besides letters. Learn more about symbols [here](#).

The labels' **default style** is `NULL` so they inherit the parent's style.

5.16.2 Style usage

- Use all properties from `style.text`
- For background drawing `style.body` properties are used

5.16.3 Notes

The label's **click enable attribute is disabled** by default. You can enable clicking with `lv_obj_set_click(label, true)`

5.16.4 Example

Title Label

Align lines to the middle

Re-color words of the text

If a line become too long it
can be automatically broken
into multiple lines

Example of Label in LittlevGL

Graphics Library

```
/*Create label on the screen. By default it will inherit the style of the screen*/
lv_obj_t * title = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(title, "Title Label");
lv_obj_align(title, NULL, LV_ALIGN_IN_TOP_MID, 0, 20); /*Align to the top*/

/*Create anew style*/
static lv_style_t style_txt;
lv_style_copy(&style_txt, &lv_style_plain);
style_txt.text.font = &lv_font_dejavu_40;
style_txt.text.letter_space = 2;
style_txt.text.line_space = 1;
style_txt.text.color = LV_COLOR_HEX(0x606060);

/*Create a new label*/
lv_obj_t * txt = lv_label_create(lv_scr_act(), NULL);
lv_obj_set_style(txt, &style_txt); /*Set the created style*/
lv_label_set_long_mode(txt, LV_LABEL_LONG_BREAK); /*Break the long lines*/
lv_label_set_recolor(txt, true); /*Enable re-coloring by
↳commands in the text*/
lv_label_set_align(txt, LV_LABEL_ALIGN_CENTER); /*Center aligned lines*/
lv_label_set_text(txt, "Align lines to the middle\n\n"
"#000080 Re-color# #0000ff words of# #6666ff the text#\n\n"
"If a line become too long it can be automatically broken into
↳multiple lines");
lv_obj_set_width(txt, 300); /*Set a width*/
lv_obj_align(txt, NULL, LV_ALIGN_CENTER, 0, 20); /*Align to center*/
```

5.17 LED (lv_led)

5.17.1 Overview

The LEDs are rectangle-like (or circle) object. You can set their **brightness** with `lv_led_set_bright(led, bright)`. The brightness should be between 0 (darkest) and 255 (lightest).

Use `lv_led_on(led)` and `lv_led_off(led)` to set the brightness to a predefined ON or OFF value. The `lv_led_toggle(led)` toggles between the ON and OFF state.

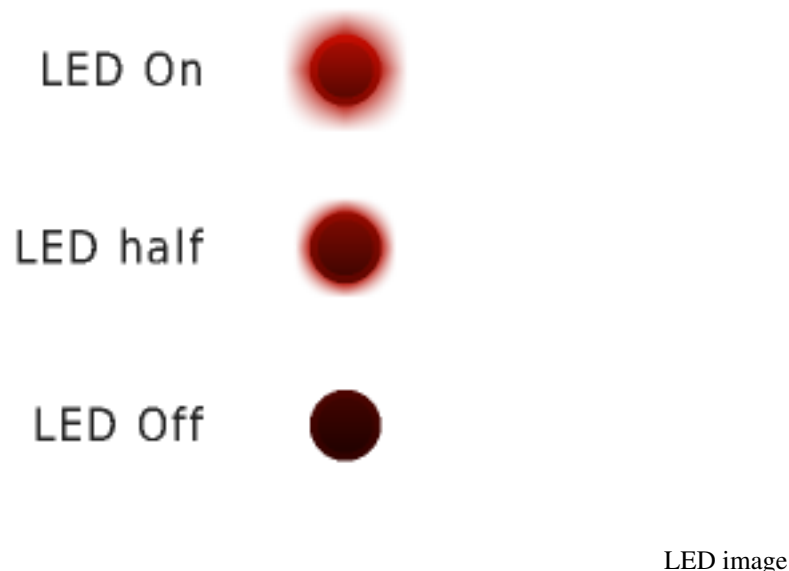
5.17.2 Style usage

The LED uses one style which can be set by `lv_led_set_style(led, &style)`. To determine the appearance the **style.body** properties are used. The colors are darkened and shadow width is reduced at a lower brightness and gains its original value at brightness 255 to show a lighting effect. The default style is: `lv_style_pretty_color`.

5.17.3 Notes

- Typically the default style is not suitable therefore you have to create your own style. See the Examples.

5.17.4 Example



```
/*Create a style for the LED*/
static lv_style_t style_led;
lv_style_copy(&style_led, &lv_style_pretty_color);
style_led.body.radius = LV_RADIUS_CIRCLE;
style_led.body.main_color = LV_COLOR_MAKE(0xb5, 0x0f, 0x04);
style_led.body.grad_color = LV_COLOR_MAKE(0x50, 0x07, 0x02);
style_led.body.border.color = LV_COLOR_MAKE(0xfa, 0x0f, 0x00);
```

(continues on next page)

(continued from previous page)

```

style_led.body.border.width = 3;
style_led.body.border.opa = LV_OPA_30;
style_led.body.shadow.color = LV_COLOR_MAKE(0xb5, 0x0f, 0x04);
style_led.body.shadow.width = 10;

/*Create a LED and switch it ON*/
lv_obj_t * led1 = lv_led_create(lv_scr_act(), NULL);
lv_obj_set_style(led1, &style_led);
lv_obj_align(led1, NULL, LV_ALIGN_IN_TOP_MID, 40, 40);
lv_led_on(led1);

/*Copy the previous LED and set a brightness*/
lv_obj_t * led2 = lv_led_create(lv_scr_act(), led1);
lv_obj_align(led2, led1, LV_ALIGN_OUT_BOTTOM_MID, 0, 40);
lv_led_set_bright(led2, 190);

/*Copy the previous LED and switch it OFF*/
lv_obj_t * led3 = lv_led_create(lv_scr_act(), led1);
lv_obj_align(led3, led2, LV_ALIGN_OUT_BOTTOM_MID, 0, 40);
lv_led_off(led3);

/*Create 3 labels next to the LEDs*/
lv_obj_t * label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label, "LED On");
lv_obj_align(label, led1, LV_ALIGN_OUT_LEFT_MID, -40, 0);

label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label, "LED half");
lv_obj_align(label, led2, LV_ALIGN_OUT_LEFT_MID, -40, 0);

label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label, "LED Off");
lv_obj_align(label, led3, LV_ALIGN_OUT_LEFT_MID, -40, 0);

```

5.18 Line (lv_line)

5.18.1 Overview

The line object is capable of **drawing straight lines** between a set of points. The points has to be stored in an `lv_point_t` array and passed to the object by the `lv_line_set_points(lines, point_array, point_num)` function.

It is possible to **automatically set the size** of the line object according to its points. You can enable it with the `lv_line_set_auto_size(line, true)` function. If enabled then when the points are set then the object width and height will be changed according to the max. x and max. y coordinates among the points. The *auto size* is enabled by default.

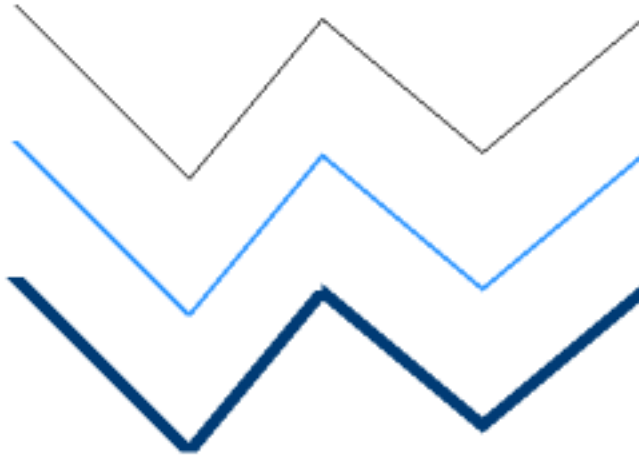
Basically the `y == 0` point is in the top of the object but you can **invert the y coordinates** with `lv_line_set_y_invert(line, true)`. After it the y coordinates will be subtracted from object's height.

5.18.2 Style usage

- `style.line` properties are used

5.18.3 Notes

5.18.4 Example



Example of Line in LittlevGL Graphics Library

ics Library

```
/*Create an array for the points of the line*/
static lv_point_t line_points[] = { {5, 5}, {70, 70}, {120, 10}, {180, 60}, {240, 10} };

/*Create line with default style*/
lv_obj_t * line1;
line1 = lv_line_create(lv_scr_act(), NULL);
lv_line_set_points(line1, line_points, 5); /*Set the points*/
lv_obj_align(line1, NULL, LV_ALIGN_IN_TOP_MID, 0, 20);

/*Create new style (thin light blue)*/
static lv_style_t style_line2;
lv_style_copy(&style_line2, &lv_style_plain);
style_line2.line.color = LV_COLOR_MAKE(0x2e, 0x96, 0xff);
style_line2.line.width = 2;

/*Copy the previous line and apply the new style*/
lv_obj_t * line2 = lv_line_create(lv_scr_act(), line1);
lv_line_set_style(line2, &style_line2);
lv_obj_align(line2, line1, LV_ALIGN_OUT_BOTTOM_MID, 0, -20);

/*Create new style (thick dark blue)*/
static lv_style_t style_line3;
lv_style_copy(&style_line3, &lv_style_plain);
```

(continues on next page)

(continued from previous page)

```
style_line3.line.color = LV_COLOR_MAKE(0x00, 0x3b, 0x75);
style_line3.line.width = 5;

/*Copy the previous line and apply the new style*/
lv_obj_t * line3 = lv_line_create(lv_scr_act(), line1);
lv_line_set_style(line3, &style_line3);
lv_obj_align(line3, line2, LV_ALIGN_OUT_BOTTOM_MID, 0, -20);
```

5.19 List (lv_list)

5.19.1 Overview

The Lists are built from a background **Page** and **Buttons** on it. The Buttons contain an optional icon-like Image (which can be a symbol too) and a Label. When the list become long enough it can be scrolled. The **width of the buttons** is set to maximum according to the object width. The **height** of the buttons are adjusted automatically according to the content (content height + style.body.padding.ver).

You can **add new list element** with `lv_list_add(list, "U:/img", "Text", rel_action)` or with symbol icon `lv_list_add(list, SYMBOL_EDIT, "Edit text")`. If you do not want to add image use NULL as file name. The function returns with a pointer to the created button to allow further configurations.

You can use `lv_list_get_btn_label(list_btn)` and `lv_list_get_btn_img(list_btn)` to **get the label and the image** of a list button.

In the release action of a button you can get the **button's text** with `lv_list_get_btn_text(button)`. It helps to identify the released list element.

To **delete a list element** just use `lv_obj_del()` on the return value of `lv_list_add()`.

You can **navigate manually** in the list with `lv_list_up(list)` and `lv_list_down(list)`.

You can focus on a button directly using `lv_list_focus(btn, anim_en)`.

The **animation time** of up/down/focus movements can be set via: `lv_list_set_anim_time(list, anim_time)`. Zero animation time means not animations.

5.19.2 Style usage

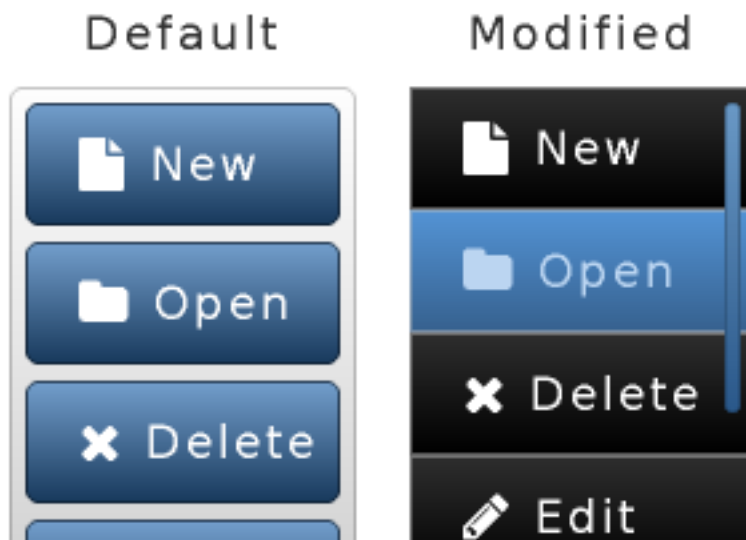
The `lv_list_set_style(list, LV_LIST_STYLE_..., &style)` function sets the style of a list. For details explanation of *BG*, *SCRL* and *SB* see [Page](#)

- **LV_LIST_STYLE_BG** list background style. Default: *lv_style_transp_fit*
- **LV_LIST_STYLE_SCRL** scrollable parts's style. Default: *lv_style_pretty_*
- **LV_LIST_STYLE_SB** scrollbars' style. Default: *lv_style_pretty_color*
- **LV_LIST_STYLE_BTN_REL** button released style. Default: *lv_style_btn_rel*
- **LV_LIST_STYLE_BTN_PR** button pressed style. Default: *lv_style_btn_pr*
- **LV_LIST_STYLE_BTN_TGL_REL** button toggled released style. Default: *lv_style_btn_tgl_rel*
- **LV_LIST_STYLE_BTN_TGL_PR** button toggled pressed style. Default: *lv_style_btn_tgl_pr*
- **LV_LIST_STYLE_BTN_INA** button inactive style. Default: *lv_style_btn_ina*

5.19.3 Notes

- You can set a transparent background for the list. In this case if you have only a few list buttons the the list will look shorter but become scrollable when more list elements are added.
- The button labels default long mode is `LV_LABEL_LONG_ROLL`. You can modify it manually. Use `lv_list_get_btn_label()` to get buttons's label.
- To **modify the height of the buttons** adjust the *body.padding.ver* field of the corresponding style (`LV_LIST_STYLE_BTN_REL`, `LV_LIST_STYLE_BTN_PR` etc.)

5.19.4 Example



List image

```

/*Will be called on click of a button of a list*/
static lv_res_t list_release_action(lv_obj_t * list_btn)
{
    printf("List element click:%s\n", lv_list_get_btn_text(list_btn));

    return LV_RES_OK; /*Return OK because the list is not deleted*/
}

.
.
.

/*****
 * Create a default list
 *****/

/*Crte the list*/
lv_obj_t * list1 = lv_list_create(lv_scr_act(), NULL);
lv_obj_set_size(list1, 130, 170);
lv_obj_align(list1, NULL, LV_ALIGN_IN_TOP_LEFT, 20, 40);

```

(continues on next page)

(continued from previous page)

```

/*Add list elements*/
lv_list_add(list1, SYMBOL_FILE, "New", list_release_action);
lv_list_add(list1, SYMBOL_DIRECTORY, "Open", list_release_action);
lv_list_add(list1, SYMBOL_CLOSE, "Delete", list_release_action);
lv_list_add(list1, SYMBOL_EDIT, "Edit", list_release_action);
lv_list_add(list1, SYMBOL_SAVE, "Save", list_release_action);

/*Create a label above the list*/
lv_obj_t * label;
label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label, "Default");
lv_obj_align(label, list1, LV_ALIGN_OUT_TOP_MID, 0, -10);

/*****
 * Create new styles
 *****/
/*Create a scroll bar style*/
static lv_style_t style_sb;
lv_style_copy(&style_sb, &lv_style_plain);
style_sb.body.main_color = LV_COLOR_BLACK;
style_sb.body.grad_color = LV_COLOR_BLACK;
style_sb.body.border.color = LV_COLOR_WHITE;
style_sb.body.border.width = 1;
style_sb.body.border.opa = LV_OPA_70;
style_sb.body.radius = LV_RADIUS_CIRCLE;
style_sb.body.opa = LV_OPA_60;

/*Create styles for the buttons*/
static lv_style_t style_btn_rel;
static lv_style_t style_btn_pr;
lv_style_copy(&style_btn_rel, &lv_style_btn_rel);
style_btn_rel.body.main_color = LV_COLOR_MAKE(0x30, 0x30, 0x30);
style_btn_rel.body.grad_color = LV_COLOR_BLACK;
style_btn_rel.body.border.color = LV_COLOR_SILVER;
style_btn_rel.body.border.width = 1;
style_btn_rel.body.border.opa = LV_OPA_50;
style_btn_rel.body.radius = 0;

lv_style_copy(&style_btn_pr, &style_btn_rel);
style_btn_pr.body.main_color = LV_COLOR_MAKE(0x55, 0x96, 0xd8);
style_btn_pr.body.grad_color = LV_COLOR_MAKE(0x37, 0x62, 0x90);
style_btn_pr.text_color = LV_COLOR_MAKE(0xbb, 0xd5, 0xf1);

/*****
 * Create a list with modified styles
 *****/

/*Copy the previous list*/
lv_obj_t * list2 = lv_list_create(lv_scr_act(), list1);
lv_obj_align(list2, NULL, LV_ALIGN_IN_TOP_RIGHT, -20, 40);
lv_list_set_sb_mode(list2, LV_SB_MODE_AUTO);
lv_list_set_style(list2, LV_LIST_STYLE_BG, &lv_style_transp_tight);
lv_list_set_style(list2, LV_LIST_STYLE_SCRL, &lv_style_transp_tight);
lv_list_set_style(list2, LV_LIST_STYLE_BTN_REL, &style_btn_rel); /*Set the new button_
↪ styles*/
lv_list_set_style(list2, LV_LIST_STYLE_BTN_PR, &style_btn_pr);

```

(continues on next page)

(continued from previous page)

```
/*Create a label above the list*/
label = lv_label_create(lv_scr_act(), label);          /*Copy the previous label*/
lv_label_set_text(label, "Modified");
lv_obj_align(label, list2, LV_ALIGN_OUT_TOP_MID, 0, -10);
```

5.20 Line meter (lv_lmeter)

5.20.1 Overview

The Line Meter object consists of some **radial lines** which draw a scale. When setting a new value with `lv_lmeter_set_value(lmeter, new_value)` the proportional part of the scale will be recolored.

The `lv_lmeter_set_range(lmeter, min, max)` function sets the **range** of the line meter.

You can set the **angle** of the scale and the **number of the lines** by: `lv_lmeter_set_scale(lmeter, angle, line_num)`. The default angle is 240 and the default line number is 31.

5.20.2 Styles

The line meter uses one style which can be set by `lv_lmeter_set_style(lmeter, LV_LMETER_STYLE_MAIN, &style)`. The line meter's properties are derived from the following style attributes:

- **line.color** “inactive line’s” color which are greater then the current value
- **body.main_color** “active line’s” color at the beginning of the scale
- **body.grad_color** “active line’s” color at the end of the scale (gradient with main color)
- **body.padding.hor** line length
- **line.width** line width

The default style is `lv_style_pretty_color`.

5.20.3 Events

Only the [Generic events](#) are sent by the object type.

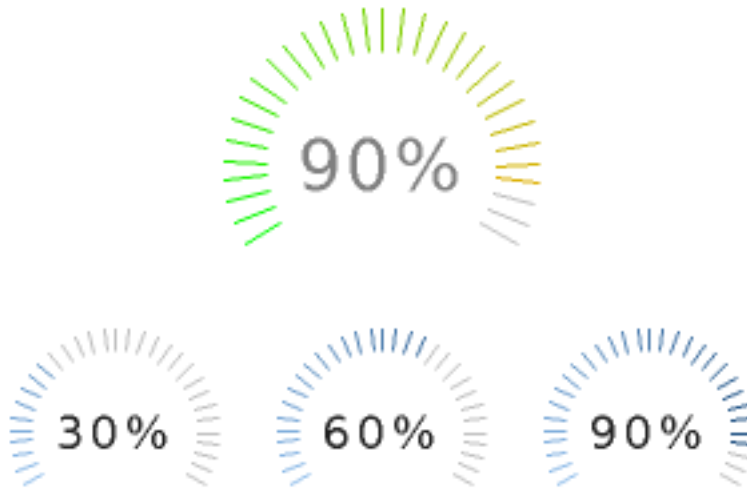
Learn more about [Events](#).

5.20.4 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.20.5 Example



Line meter image

```

/*****
 * Create 3 similar line meter
 *****/

/*Create a simple style with ticker line width*/
static lv_style_t style_lmeter1;
lv_style_copy(&style_lmeter1, &lv_style_pretty_color);
style_lmeter1.line.width = 2;
style_lmeter1.line.color = LV_COLOR_SILVER;
style_lmeter1.body.main_color = LV_COLOR_HEX(0x91bfed);           /*Light blue*/
style_lmeter1.body.grad_color = LV_COLOR_HEX(0x04386c);          /*Dark blue*/

/*Create the first line meter */
lv_obj_t * lmeter;
lmeter = lv_lmeter_create(lv_scr_act(), NULL);
lv_lmeter_set_range(lmeter, 0, 100);                               /*Set the range*/
lv_lmeter_set_value(lmeter, 30);                                   /*Set the current value*/
lv_lmeter_set_style(lmeter, &style_lmeter1);                       /*Apply the new style*/
lv_obj_set_size(lmeter, 80, 80);
lv_obj_align(lmeter, NULL, LV_ALIGN_IN_BOTTOM_LEFT, 20, -20);

/*Add a label to show the current value*/
lv_obj_t * label;
label = lv_label_create(lmeter, NULL);
lv_label_set_text(label, "30%");
lv_label_set_style(label, &lv_style_pretty);
lv_obj_align(label, NULL, LV_ALIGN_CENTER, 0, 0);

/*Create the second line meter and label*/
lmeter = lv_lmeter_create(lv_scr_act(), lmeter);
lv_lmeter_set_value(lmeter, 60);
lv_obj_align(lmeter, NULL, LV_ALIGN_IN_BOTTOM_MID, 0, -20);

```

(continues on next page)

(continued from previous page)

```

label = lv_label_create(lmeter, label);
lv_label_set_text(label, "60%");
lv_obj_align(label, NULL, LV_ALIGN_CENTER, 0, 0);

/*Create the third line meter and label*/
lmeter = lv_lmeter_create(lv_scr_act(), lmeter);
lv_lmeter_set_value(lmeter, 90);
lv_obj_align(lmeter, NULL, LV_ALIGN_IN_BOTTOM_RIGHT, -20, -20);

label = lv_label_create(lmeter, label);
lv_label_set_text(label, "90%");
lv_obj_align(label, NULL, LV_ALIGN_CENTER, 0, 0);

/*****
 * Create a greater line meter
 *****/

/*Create a new style*/
static lv_style_t style_lmeter2;
lv_style_copy(&style_lmeter2, &lv_style_pretty_color);
style_lmeter2.line.width = 2;
style_lmeter2.line.color = LV_COLOR_SILVER;
style_lmeter2.body.padding.hor = 16; /*Line length*/
style_lmeter2.body.main_color = LV_COLOR_LIME;
style_lmeter2.body.grad_color = LV_COLOR_ORANGE;

/*Create the line meter*/
lmeter = lv_lmeter_create(lv_scr_act(), lmeter);
lv_obj_set_style(lmeter, &style_lmeter2);
lv_obj_set_size(lmeter, 120, 120);
lv_obj_align(lmeter, NULL, LV_ALIGN_IN_TOP_MID, 0, 20);
lv_lmeter_set_scale(lmeter, 240, 31);
lv_lmeter_set_value(lmeter, 90);

/*Create a label style with greater font*/
static lv_style_t style_label;
lv_style_copy(&style_label, &lv_style_pretty);
style_label.text.font = &lv_font_dejavu_60;
style_label.text.color = LV_COLOR_GRAY;

/*Add a label to show the current value*/
label = lv_label_create(lmeter, label);
lv_label_set_text(label, "90%");
lv_obj_set_style(label, &style_label);
lv_obj_align(label, NULL, LV_ALIGN_CENTER, 0, 0);

```

5.21 Message box (lv_mbox)

5.22 Page (lv_page)

5.23 Preload (lv_preload)

5.24 Roller (lv_roller)

5.25 Slider (lv_slider)

5.25.1 Overview

The Slider object looks like a *Bar* supplemented **with a knob**. The knob can be **dragged to set a value**. The Slider also can be vertical or horizontal.

`lv_slider_set_anim_time(slider, anim_time)` sets the animation time in milliseconds.

To set an **initial value** use `lv_slider_set_value(slider, new_value, LV_ANIM_ON/OFF)`.

To specify the **range** (min, max values) the `lv_slider_set_range(slider, min, max)` can be used.

The **knob can be placed** two ways:

- inside the background
- on the edges on min/max values

Use the `lv_slider_set_knob_in(slider, true/false)` to choose between the modes. (*knob_in = false* is the default)

5.25.2 Styles

You can modify the slider's styles with `lv_slider_set_style(slider, LV_SLIDER_STYLE_..., &style)`.

- **LV_SLIDER_STYLE_BG** Style of the background. All `style.body` properties are used. The padding values make the knob larger than background. (negative value makes is larger)
- **LV_SLIDER_STYLE_INDIC** Style of the indicator. All `style.body` properties are used. The padding values make the indicator smaller than the background.
- **LV_SLIDER_STYLE_KNOB** Style of the knob. All `style.body` properties are used except padding.

5.25.3 Events

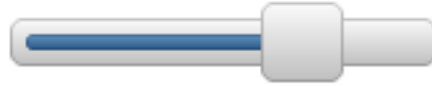
- **LV_EVENT_VALUE_CHANGED** Sent while slider is being dragged or changed with keys.

5.25.4 Keys

- **LV_KEY_UP, LV_KEY_RIGHT** Increment the slider's value by 1
- **LV_KEY_DOWN, LV_KEY_LEFT** Decrement the slider's value by 1

5.25.5 Example

Default



Modified



Slider image

```

/*Called when a new value is set on the slider*/
static lv_res_t slider_action(lv_obj_t * slider)
{
    printf("New slider value: %d\n", lv_slider_get_value(slider));

    return LV_RES_OK;
}

.
.
.

/*Create a default slider*/
lv_obj_t * slider1 = lv_slider_create(lv_scr_act(), NULL);
lv_obj_set_size(slider1, 160, 30);
lv_obj_align(slider1, NULL, LV_ALIGN_IN_TOP_RIGHT, -30, 30);
lv_slider_set_action(slider1, slider_action);
lv_bar_set_value(slider1, 70);

/*Create a label right to the slider*/
lv_obj_t * slider1_label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(slider1_label, "Default");
lv_obj_align(slider1_label, slider1, LV_ALIGN_OUT_LEFT_MID, -20, 0);

/*Create a bar, an indicator and a knob style*/
static lv_style_t style_bg;
static lv_style_t style_indic;
static lv_style_t style_knob;

lv_style_copy(&style_bg, &lv_style_pretty);
style_bg.body.main_color = LV_COLOR_BLACK;
style_bg.body.grad_color = LV_COLOR_GRAY;
    
```

(continues on next page)

(continued from previous page)

```

style_bg.body.radius = LV_RADIUS_CIRCLE;
style_bg.body.border.color = LV_COLOR_WHITE;

lv_style_copy(&style_indic, &lv_style_pretty);
style_indic.body.grad_color = LV_COLOR_GREEN;
style_indic.body.main_color = LV_COLOR_LIME;
style_indic.body.radius = LV_RADIUS_CIRCLE;
style_indic.body.shadow.width = 10;
style_indic.body.shadow.color = LV_COLOR_LIME;
style_indic.body.padding.hor = 3;
style_indic.body.padding.ver = 3;

lv_style_copy(&style_knob, &lv_style_pretty);
style_knob.body.radius = LV_RADIUS_CIRCLE;
style_knob.body.opa = LV_OPA_70;
style_knob.body.padding.ver = 10 ;

/*Create a second slider*/
lv_obj_t * slider2 = lv_slider_create(lv_scr_act(), slider1);
lv_slider_set_style(slider2, LV_SLIDER_STYLE_BG, &style_bg);
lv_slider_set_style(slider2, LV_SLIDER_STYLE_INDIC, &style_indic);
lv_slider_set_style(slider2, LV_SLIDER_STYLE_KNOB, &style_knob);
lv_obj_align(slider2, slider1, LV_ALIGN_OUT_BOTTOM_MID, 0, 30); /*Align below 'bar1'*/

/*Create a second label*/
lv_obj_t * slider2_label = lv_label_create(lv_scr_act(), slider1_label);
lv_label_set_text(slider2_label, "Modified");
lv_obj_align(slider2_label, slider2, LV_ALIGN_OUT_LEFT_MID, -30, 0);

```

5.26 Spinbox (lv_spinbox)

5.27 Switch (lv_sw)

5.27.1 Overview

The Switch can be used to **turn on/off** something. The look like a little slider. The state of the switch can be changed by:

- Clicking on it
- Sliding it
- Using `lv_sw_on(sw)` and `lv_sw_off(sw)` functions

A **callback function** can be assigned to call when the user uses the switch: `lv_sw_set_action(sw, my_action)`.

New in v5.3: Switches can be animated by calling `lv_sw_set_anim_time(sw, anim_ms)`.

5.27.2 Style usage

You can modify the Switch's styles with `lv_sw_set_style(sw, LV_SW_STYLE_..., &style)`.

- **LV_SW_STYLE_BG** Style of the background. All *style.body* properties are used. The *padding* values make the Switch smaller then the knob. (negative value makes is larger)
- **LV_SW_STYLE_INDIC** Style of the indicator. All *style.body* properties are used. The *padding* values make the indicator smaller then the background.
- **LV_SW_STYLE_KNOB_OFF** Style of the knob when the switch is off. The *style.body* properties are used except padding.
- **LV_SW_STYLE_KNOB_ON** Style of the knob when the switch is on. The *style.body* properties are used except padding.

5.27.3 Notes

- The Knob is not a real object it is only drawn above the Bar

5.27.4 Example



Switch image

```
/*Create styles for the switch*/
static lv_style_t bg_style;
static lv_style_t indic_style;
static lv_style_t knob_on_style;
static lv_style_t knob_off_style;
lv_style_copy(&bg_style, &lv_style_pretty);
bg_style.body.radius = LV_RADIUS_CIRCLE;

lv_style_copy(&indic_style, &lv_style_pretty_color);
indic_style.body.radius = LV_RADIUS_CIRCLE;
indic_style.body.main_color = LV_COLOR_HEX(0x9fc8ef);
indic_style.body.grad_color = LV_COLOR_HEX(0x9fc8ef);
indic_style.body.padding.hor = 0;
indic_style.body.padding.ver = 0;
```

(continues on next page)

(continued from previous page)

```
lv_style_copy(&knob_off_style, &lv_style_pretty);
knob_off_style.body.radius = LV_RADIUS_CIRCLE;
knob_off_style.body.shadow.width = 4;
knob_off_style.body.shadow.type = LV_SHADOW_BOTTOM;

lv_style_copy(&knob_on_style, &lv_style_pretty_color);
knob_on_style.body.radius = LV_RADIUS_CIRCLE;
knob_on_style.body.shadow.width = 4;
knob_on_style.body.shadow.type = LV_SHADOW_BOTTOM;

/*Create a switch and apply the styles*/
lv_obj_t *sw1 = lv_sw_create(lv_scr_act(), NULL);
lv_sw_set_style(sw1, LV_SW_STYLE_BG, &bg_style);
lv_sw_set_style(sw1, LV_SW_STYLE_INDIC, &indic_style);
lv_sw_set_style(sw1, LV_SW_STYLE_KNOB_ON, &knob_on_style);
lv_sw_set_style(sw1, LV_SW_STYLE_KNOB_OFF, &knob_off_style);
lv_obj_align(sw1, NULL, LV_ALIGN_CENTER, 0, -50);

/*Copy the first switch and turn it ON*/
lv_obj_t *sw2 = lv_sw_create(lv_scr_act(), sw1);
lv_sw_on(sw2);
lv_obj_align(sw2, NULL, LV_ALIGN_CENTER, 0, 50);
```

5.28 Table (lv_table)

5.29 Tab view (lv_tabview)

5.30 Text area (lv_ta)

5.31 Tile view (lv_tileview)

5.32 Window (lv_win)

Symbols

[anonymous] (C++ *enum*), 45, 49, 54, 62, 69, 75, 82, 88, 97, 102, 108, 114, 119, 124

L

lv_arc_create (C++ *function*), 45
 lv_arc_ext_t (C++ *class*), 46
 lv_arc_ext_t::angle_end (C++ *member*), 46
 lv_arc_ext_t::angle_start (C++ *member*), 46
 lv_arc_get_angle_end (C++ *function*), 45
 lv_arc_get_angle_start (C++ *function*), 45
 lv_arc_get_style (C++ *function*), 46
 lv_arc_set_angles (C++ *function*), 45
 lv_arc_set_style (C++ *function*), 45
 LV_ARC_STYLE_MAIN (C++ *enumerator*), 45
 lv_arc_style_t (C++ *type*), 45
 LV_BAR_ANIM_STATE_END (C *macro*), 48
 LV_BAR_ANIM_STATE_INV (C *macro*), 48
 LV_BAR_ANIM_STATE_NORM (C *macro*), 48
 LV_BAR_ANIM_STATE_START (C *macro*), 48
 lv_bar_create (C++ *function*), 49
 lv_bar_ext_t (C++ *class*), 51
 lv_bar_ext_t::anim_end (C++ *member*), 51
 lv_bar_ext_t::anim_start (C++ *member*), 51
 lv_bar_ext_t::anim_state (C++ *member*), 51
 lv_bar_ext_t::anim_time (C++ *member*), 51
 lv_bar_ext_t::cur_value (C++ *member*), 51
 lv_bar_ext_t::max_value (C++ *member*), 51
 lv_bar_ext_t::min_value (C++ *member*), 51
 lv_bar_ext_t::style_indic (C++ *member*), 51
 lv_bar_ext_t::sym (C++ *member*), 51
 lv_bar_get_anim_time (C++ *function*), 50
 lv_bar_get_max_value (C++ *function*), 50
 lv_bar_get_min_value (C++ *function*), 50
 lv_bar_get_style (C++ *function*), 50
 lv_bar_get_sym (C++ *function*), 50
 lv_bar_get_value (C++ *function*), 50
 lv_bar_set_anim_time (C++ *function*), 49
 lv_bar_set_range (C++ *function*), 49
 lv_bar_set_style (C++ *function*), 50
 lv_bar_set_sym (C++ *function*), 49
 lv_bar_set_value (C++ *function*), 49

LV_BAR_STYLE_BG (C++ *enumerator*), 49
 LV_BAR_STYLE_INDIC (C++ *enumerator*), 49
 lv_bar_style_t (C++ *type*), 48
 lv_btn_create (C++ *function*), 55
 lv_btn_ext_t (C++ *class*), 58
 lv_btn_ext_t::cont (C++ *member*), 58
 lv_btn_ext_t::ink_in_time (C++ *member*), 58
 lv_btn_ext_t::ink_out_time (C++ *member*), 58
 lv_btn_ext_t::ink_wait_time (C++ *member*), 58
 lv_btn_ext_t::state (C++ *member*), 58
 lv_btn_ext_t::styles (C++ *member*), 58
 lv_btn_ext_t::toggle (C++ *member*), 58
 lv_btn_get_fit_bottom (C++ *function*), 57
 lv_btn_get_fit_left (C++ *function*), 57
 lv_btn_get_fit_right (C++ *function*), 57
 lv_btn_get_fit_top (C++ *function*), 57
 lv_btn_get_ink_in_time (C++ *function*), 57
 lv_btn_get_ink_out_time (C++ *function*), 58
 lv_btn_get_ink_wait_time (C++ *function*), 58
 lv_btn_get_layout (C++ *function*), 57
 lv_btn_get_state (C++ *function*), 57
 lv_btn_get_style (C++ *function*), 58
 lv_btn_get_toggle (C++ *function*), 57
 lv_btn_set_fit (C++ *function*), 56
 lv_btn_set_fit2 (C++ *function*), 56
 lv_btn_set_fit4 (C++ *function*), 55
 lv_btn_set_ink_in_time (C++ *function*), 56
 lv_btn_set_ink_out_time (C++ *function*), 56
 lv_btn_set_ink_wait_time (C++ *function*), 56
 lv_btn_set_layout (C++ *function*), 55
 lv_btn_set_state (C++ *function*), 55
 lv_btn_set_style (C++ *function*), 56
 lv_btn_set_toggle (C++ *function*), 55
 LV_BTN_STATE_INA (C++ *enumerator*), 54
 LV_BTN_STATE_NUM (C++ *enumerator*), 54
 LV_BTN_STATE_PR (C++ *enumerator*), 54
 LV_BTN_STATE_REL (C++ *enumerator*), 54
 lv_btn_state_t (C++ *type*), 54
 LV_BTN_STATE_TGL_PR (C++ *enumerator*), 54
 LV_BTN_STATE_TGL_REL (C++ *enumerator*), 54

[LV_BTN_STYLE_INA \(C++ enumerator\), 55](#)
[LV_BTN_STYLE_PR \(C++ enumerator\), 54](#)
[LV_BTN_STYLE_REL \(C++ enumerator\), 54](#)
[lv_btn_style_t \(C++ type\), 54](#)
[LV_BTN_STYLE_TGL_PR \(C++ enumerator\), 55](#)
[LV_BTN_STYLE_TGL_REL \(C++ enumerator\), 54](#)
[lv_btn_toggle \(C++ function\), 55](#)
[LV_BTNM_BTN_NONE \(C macro\), 62](#)
[lv_btnm_clear_btn_ctrl \(C++ function\), 64](#)
[lv_btnm_clear_btn_ctrl_all \(C++ function\), 64](#)
[lv_btnm_create \(C++ function\), 63](#)
[LV_BTNM_CTRL_CLICK_TRIG \(C++ enumerator\), 62](#)
[LV_BTNM_CTRL_HIDDEN \(C++ enumerator\), 62](#)
[LV_BTNM_CTRL_INACTIVE \(C++ enumerator\), 62](#)
[LV_BTNM_CTRL_NO_REPEAT \(C++ enumerator\), 62](#)
[lv_btnm_ctrl_t \(C++ type\), 62](#)
[LV_BTNM_CTRL_TGL_ENABLE \(C++ enumerator\), 62](#)
[LV_BTNM_CTRL_TGL_STATE \(C++ enumerator\), 62](#)
[lv_btnm_ext_t \(C++ class\), 66](#)
[lv_btnm_ext_t::btn_cnt \(C++ member\), 66](#)
[lv_btnm_ext_t::btn_id_act \(C++ member\), 66](#)
[lv_btnm_ext_t::btn_id_pr \(C++ member\), 66](#)
[lv_btnm_ext_t::button_areas \(C++ member\), 66](#)
[lv_btnm_ext_t::ctrl_bits \(C++ member\), 66](#)
[lv_btnm_ext_t::map_p \(C++ member\), 66](#)
[lv_btnm_ext_t::one_toggle \(C++ member\), 66](#)
[lv_btnm_ext_t::recolor \(C++ member\), 66](#)
[lv_btnm_ext_t::styles_btn \(C++ member\), 66](#)
[lv_btnm_get_active_btn \(C++ function\), 65](#)
[lv_btnm_get_active_btn_text \(C++ function\), 65](#)
[lv_btnm_get_btn_ctrl \(C++ function\), 65](#)
[lv_btnm_get_btn_text \(C++ function\), 65](#)
[lv_btnm_get_map_array \(C++ function\), 64](#)
[lv_btnm_get_one_toggle \(C++ function\), 66](#)
[lv_btnm_get_pressed_btn \(C++ function\), 65](#)
[lv_btnm_get_recolor \(C++ function\), 65](#)
[lv_btnm_get_style \(C++ function\), 65](#)
[lv_btnm_set_btn_ctrl \(C++ function\), 63](#)
[lv_btnm_set_btn_ctrl_all \(C++ function\), 64](#)
[lv_btnm_set_btn_width \(C++ function\), 64](#)
[lv_btnm_set_ctrl_map \(C++ function\), 63](#)
[lv_btnm_set_map \(C++ function\), 63](#)
[lv_btnm_set_one_toggle \(C++ function\), 64](#)
[lv_btnm_set_pressed \(C++ function\), 63](#)
[lv_btnm_set_recolor \(C++ function\), 63](#)
[lv_btnm_set_style \(C++ function\), 63](#)
[LV_BTNM_STYLE_BG \(C++ enumerator\), 62](#)
[LV_BTNM_STYLE_BTN_INA \(C++ enumerator\), 62](#)
[LV_BTNM_STYLE_BTN_PR \(C++ enumerator\), 62](#)
[LV_BTNM_STYLE_BTN_REL \(C++ enumerator\), 62](#)
[LV_BTNM_STYLE_BTN_TGL_PR \(C++ enumerator\), 62](#)
[LV_BTNM_STYLE_BTN_TGL_REL \(C++ enumerator\), 62](#)
[lv_btnm_style_t \(C++ type\), 62](#)
[LV_BTNM_WIDTH_MASK \(C macro\), 62](#)
[lv_calendar_create \(C++ function\), 70](#)
[lv_calendar_date_t \(C++ class\), 72](#)
[lv_calendar_date_t::day \(C++ member\), 72](#)
[lv_calendar_date_t::month \(C++ member\), 72](#)
[lv_calendar_date_t::year \(C++ member\), 72](#)
[lv_calendar_ext_t \(C++ class\), 72](#)
[lv_calendar_ext_t::btn_pressing \(C++ member\), 72](#)
[lv_calendar_ext_t::day_names \(C++ member\), 72](#)
[lv_calendar_ext_t::highlighted_dates \(C++ member\), 72](#)
[lv_calendar_ext_t::highlighted_dates_num \(C++ member\), 72](#)
[lv_calendar_ext_t::month_names \(C++ member\), 72](#)
[lv_calendar_ext_t::pressed_date \(C++ member\), 72](#)
[lv_calendar_ext_t::showed_date \(C++ member\), 72](#)
[lv_calendar_ext_t::style_day_names \(C++ member\), 73](#)
[lv_calendar_ext_t::style_header \(C++ member\), 73](#)
[lv_calendar_ext_t::style_header_pr \(C++ member\), 73](#)
[lv_calendar_ext_t::style_highlighted_days \(C++ member\), 73](#)
[lv_calendar_ext_t::style_inactive_days \(C++ member\), 73](#)
[lv_calendar_ext_t::style_today_box \(C++ member\), 73](#)
[lv_calendar_ext_t::style_week_box \(C++ member\), 73](#)
[lv_calendar_ext_t::today \(C++ member\), 72](#)
[lv_calendar_get_day_names \(C++ function\), 72](#)
[lv_calendar_get_highlighted_dates \(C++ function\), 71](#)
[lv_calendar_get_highlighted_dates_num \(C++ function\), 71](#)
[lv_calendar_get_month_names \(C++ function\), 72](#)
[lv_calendar_get_pressed_date \(C++ function\), 71](#)
[lv_calendar_get_showed_date \(C++ function\), 71](#)
[lv_calendar_get_style \(C++ function\), 72](#)

lv_calendar_get_today_date (C++ function), 71

lv_calendar_set_day_names (C++ function), 70

lv_calendar_set_highlighted_dates (C++ function), 70

lv_calendar_set_month_names (C++ function), 71

lv_calendar_set_showed_date (C++ function), 70

lv_calendar_set_style (C++ function), 71

lv_calendar_set_today_date (C++ function), 70

LV_CALENDAR_STYLE_BG (C++ enumerator), 69

LV_CALENDAR_STYLE_DAY_NAMES (C++ enumerator), 69

LV_CALENDAR_STYLE_HEADER (C++ enumerator), 69

LV_CALENDAR_STYLE_HEADER_PR (C++ enumerator), 69

LV_CALENDAR_STYLE_HEADER_PR (C++ enumerator), 69

LV_CALENDAR_STYLE_HIGHLIGHTED_DAYS (C++ enumerator), 70

LV_CALENDAR_STYLE_INACTIVE_DAYS (C++ enumerator), 70

lv_calendar_style_t (C++ type), 69

LV_CALENDAR_STYLE_TODAY_BOX (C++ enumerator), 70

LV_CALENDAR_STYLE_WEEK_BOX (C++ enumerator), 70

LV_CANVAS_BUF_SIZE_ALPHA_1BIT (C macro), 74

LV_CANVAS_BUF_SIZE_ALPHA_2BIT (C macro), 74

LV_CANVAS_BUF_SIZE_ALPHA_4BIT (C macro), 74

LV_CANVAS_BUF_SIZE_ALPHA_8BIT (C macro), 74

LV_CANVAS_BUF_SIZE_INDEXED_1BIT (C macro), 74

LV_CANVAS_BUF_SIZE_INDEXED_2BIT (C macro), 74

LV_CANVAS_BUF_SIZE_INDEXED_4BIT (C macro), 74

LV_CANVAS_BUF_SIZE_INDEXED_8BIT (C macro), 74

LV_CANVAS_BUF_SIZE_TRUE_COLOR (C macro), 74

LV_CANVAS_BUF_SIZE_TRUE_COLOR_ALPHA (C macro), 74

LV_CANVAS_BUF_SIZE_TRUE_COLOR_CHROMA_KEYED (C macro), 74

lv_canvas_copy_buf (C++ function), 77

lv_canvas_create (C++ function), 75

lv_canvas_draw_arc (C++ function), 79

lv_canvas_draw_img (C++ function), 78

lv_canvas_draw_line (C++ function), 78

lv_canvas_draw_polygon (C++ function), 78

lv_canvas_draw_rect (C++ function), 78

lv_canvas_draw_text (C++ function), 78

lv_canvas_ext_t (C++ class), 79

lv_canvas_ext_t::dsc (C++ member), 79

lv_canvas_ext_t::img (C++ member), 79

lv_canvas_fill_bg (C++ function), 77

lv_canvas_get_img (C++ function), 76

lv_canvas_get_px (C++ function), 76

lv_canvas_get_style (C++ function), 76

lv_canvas_rotate (C++ function), 77

lv_canvas_set_buffer (C++ function), 75

lv_canvas_set_palette (C++ function), 76

lv_canvas_set_px (C++ function), 75

lv_canvas_set_style (C++ function), 76

LV_CANVAS_STYLE_MAIN (C++ enumerator), 75

lv_canvas_style_t (C++ type), 75

lv_cb_create (C++ function), 83

lv_cb_ext_t (C++ class), 84

lv_cb_ext_t::bg_btn (C++ member), 84

lv_cb_ext_t::bullet (C++ member), 84

lv_cb_ext_t::label (C++ member), 84

lv_cb_get_style (C++ function), 84

lv_cb_get_text (C++ function), 83

lv_cb_is_checked (C++ function), 83

lv_cb_is_inactive (C++ function), 84

lv_cb_set_checked (C++ function), 83

lv_cb_set_inactive (C++ function), 83

lv_cb_set_static_text (C++ function), 83

lv_cb_set_style (C++ function), 83

lv_cb_set_text (C++ function), 83

LV_CB_STYLE_BG (C++ enumerator), 82

LV_CB_STYLE_BOX_INA (C++ enumerator), 82

LV_CB_STYLE_BOX_PR (C++ enumerator), 82

LV_CB_STYLE_BOX_REL (C++ enumerator), 82

LV_CB_STYLE_BOX_TGL_PR (C++ enumerator), 82

LV_CB_STYLE_BOX_TGL_REL (C++ enumerator), 82

lv_cb_style_t (C++ type), 82

lv_chart_add_series (C++ function), 88

lv_chart_axis_cfg_t (C++ class), 93

lv_chart_axis_cfg_t::list_of_values (C++ member), 93

lv_chart_axis_cfg_t::major_tick_len (C++ member), 93

lv_chart_axis_cfg_t::minor_tick_len (C++ member), 93

lv_chart_axis_cfg_t::num_tick_marks (C++ member), 93

lv_chart_axis_cfg_t::options (C++ member), 93

LV_CHART_AXIS_DRAW_LAST_TICK (C++ enumerator), 88

[lv_chart_axis_options_t \(C++ type\), 87](#)
[lv_chart_clear_serie \(C++ function\), 88](#)
[lv_chart_create \(C++ function\), 88](#)
[lv_chart_ext_t \(C++ class\), 93](#)
[lv_chart_ext_t::dark \(C++ member\), 93](#)
[lv_chart_ext_t::hdiv_cnt \(C++ member\), 93](#)
[lv_chart_ext_t::margin \(C++ member\), 93](#)
[lv_chart_ext_t::num \(C++ member\), 93](#)
[lv_chart_ext_t::opa \(C++ member\), 93](#)
[lv_chart_ext_t::point_cnt \(C++ member\), 93](#)
[lv_chart_ext_t::series \(C++ member\), 93](#)
[lv_chart_ext_t::series_ll \(C++ member\), 93](#)
[lv_chart_ext_t::type \(C++ member\), 93](#)
[lv_chart_ext_t::update_mode \(C++ member\), 93](#)
[lv_chart_ext_t::vdiv_cnt \(C++ member\), 93](#)
[lv_chart_ext_t::width \(C++ member\), 93](#)
[lv_chart_ext_t::x_axis \(C++ member\), 93](#)
[lv_chart_ext_t::y_axis \(C++ member\), 93](#)
[lv_chart_ext_t::ymax \(C++ member\), 93](#)
[lv_chart_ext_t::ymin \(C++ member\), 93](#)
[lv_chart_get_margin \(C++ function\), 92](#)
[lv_chart_get_point_cnt \(C++ function\), 91](#)
[lv_chart_get_series_darking \(C++ function\), 92](#)
[lv_chart_get_series_opa \(C++ function\), 92](#)
[lv_chart_get_series_width \(C++ function\), 92](#)
[lv_chart_get_style \(C++ function\), 92](#)
[lv_chart_get_type \(C++ function\), 91](#)
[lv_chart_init_points \(C++ function\), 89](#)
[LV_CHART_POINT_DEF \(C macro\), 87](#)
[lv_chart_refresh \(C++ function\), 92](#)
[lv_chart_series_t \(C++ class\), 92](#)
[lv_chart_series_t::color \(C++ member\), 93](#)
[lv_chart_series_t::points \(C++ member\), 93](#)
[lv_chart_series_t::start_point \(C++ member\), 93](#)
[lv_chart_set_div_line_count \(C++ function\), 88](#)
[lv_chart_set_margin \(C++ function\), 90](#)
[lv_chart_set_next \(C++ function\), 90](#)
[lv_chart_set_point_count \(C++ function\), 89](#)
[lv_chart_set_points \(C++ function\), 90](#)
[lv_chart_set_range \(C++ function\), 89](#)
[lv_chart_set_series_darking \(C++ function\), 89](#)
[lv_chart_set_series_opa \(C++ function\), 89](#)
[lv_chart_set_series_width \(C++ function\), 89](#)
[lv_chart_set_style \(C++ function\), 90](#)
[lv_chart_set_type \(C++ function\), 89](#)
[lv_chart_set_update_mode \(C++ function\), 90](#)
[lv_chart_set_x_ticks \(C++ function\), 91](#)
[lv_chart_set_y_ticks \(C++ function\), 91](#)
[LV_CHART_STYLE_MAIN \(C++ enumerator\), 88](#)
[lv_chart_style_t \(C++ type\), 87](#)
[LV_CHART_TYPE_AREA \(C++ enumerator\), 88](#)
[LV_CHART_TYPE_COLUMN \(C++ enumerator\), 88](#)
[LV_CHART_TYPE_LINE \(C++ enumerator\), 88](#)
[LV_CHART_TYPE_NONE \(C++ enumerator\), 88](#)
[LV_CHART_TYPE_POINT \(C++ enumerator\), 88](#)
[lv_chart_type_t \(C++ type\), 87](#)
[LV_CHART_TYPE_VERTICAL_LINE \(C++ enumerator\), 88](#)
[LV_CHART_UPDATE_MODE_CIRCULAR \(C++ enumerator\), 88](#)
[LV_CHART_UPDATE_MODE_SHIFT \(C++ enumerator\), 88](#)
[lv_chart_update_mode_t \(C++ type\), 87](#)
[lv_cont_create \(C++ function\), 97](#)
[lv_cont_ext_t \(C++ class\), 99](#)
[lv_cont_ext_t::fit_bottom \(C++ member\), 99](#)
[lv_cont_ext_t::fit_left \(C++ member\), 99](#)
[lv_cont_ext_t::fit_right \(C++ member\), 99](#)
[lv_cont_ext_t::fit_top \(C++ member\), 99](#)
[lv_cont_ext_t::layout \(C++ member\), 99](#)
[lv_cont_get_fit_bottom \(C++ function\), 99](#)
[lv_cont_get_fit_left \(C++ function\), 98](#)
[lv_cont_get_fit_right \(C++ function\), 98](#)
[lv_cont_get_fit_top \(C++ function\), 98](#)
[lv_cont_get_layout \(C++ function\), 98](#)
[lv_cont_get_style \(C++ function\), 99](#)
[lv_cont_set_fit \(C++ function\), 98](#)
[lv_cont_set_fit2 \(C++ function\), 98](#)
[lv_cont_set_fit4 \(C++ function\), 97](#)
[lv_cont_set_layout \(C++ function\), 97](#)
[lv_cont_set_style \(C++ function\), 98](#)
[LV_CONT_STYLE_MAIN \(C++ enumerator\), 97](#)
[lv_cont_style_t \(C++ type\), 96](#)
[lv_ddlist_close \(C++ function\), 105](#)
[lv_ddlist_create \(C++ function\), 102](#)
[lv_ddlist_ext_t \(C++ class\), 105](#)
[lv_ddlist_ext_t::anim_time \(C++ member\), 106](#)
[lv_ddlist_ext_t::draw_arrow \(C++ member\), 106](#)
[lv_ddlist_ext_t::fix_height \(C++ member\), 106](#)
[lv_ddlist_ext_t::force_sel \(C++ member\), 106](#)
[lv_ddlist_ext_t::label \(C++ member\), 106](#)
[lv_ddlist_ext_t::opened \(C++ member\), 106](#)
[lv_ddlist_ext_t::option_cnt \(C++ member\), 106](#)
[lv_ddlist_ext_t::page \(C++ member\), 106](#)
[lv_ddlist_ext_t::sel_opt_id \(C++ member\), 106](#)
[lv_ddlist_ext_t::sel_opt_id_ori \(C++ member\), 106](#)

lv_ddlist_ext_t::sel_style (C++ member), 106

lv_ddlist_ext_t::stay_open (C++ member), 106

lv_ddlist_get_align (C++ function), 105

lv_ddlist_get_anim_time (C++ function), 105

lv_ddlist_get_draw_arrow (C++ function), 104

lv_ddlist_get_fix_height (C++ function), 104

lv_ddlist_get_options (C++ function), 104

lv_ddlist_get_sb_mode (C++ function), 104

lv_ddlist_get_selected (C++ function), 104

lv_ddlist_get_selected_str (C++ function), 104

lv_ddlist_get_stay_open (C++ function), 104

lv_ddlist_get_style (C++ function), 105

lv_ddlist_open (C++ function), 105

lv_ddlist_set_align (C++ function), 104

lv_ddlist_set_anim_time (C++ function), 103

lv_ddlist_set_draw_arrow (C++ function), 103

lv_ddlist_set_fix_height (C++ function), 103

lv_ddlist_set_fix_width (C++ function), 103

lv_ddlist_set_options (C++ function), 102

lv_ddlist_set_sb_mode (C++ function), 103

lv_ddlist_set_selected (C++ function), 102

lv_ddlist_set_stay_open (C++ function), 103

lv_ddlist_set_style (C++ function), 103

LV_DDLIST_STYLE_BG (C++ enumerator), 102

LV_DDLIST_STYLE_SB (C++ enumerator), 102

LV_DDLIST_STYLE_SEL (C++ enumerator), 102

lv_ddlist_style_t (C++ type), 102

LV_FIT_FILL (C++ enumerator), 97

LV_FIT_FLOOD (C++ enumerator), 97

LV_FIT_NONE (C++ enumerator), 97

lv_fit_t (C++ type), 96

LV_FIT_TIGHT (C++ enumerator), 97

lv_gauge_create (C++ function), 108

lv_gauge_ext_t (C++ class), 111

lv_gauge_ext_t::label_count (C++ member), 111

lv_gauge_ext_t::lmeter (C++ member), 111

lv_gauge_ext_t::needle_colors (C++ member), 111

lv_gauge_ext_t::needle_count (C++ member), 111

lv_gauge_ext_t::values (C++ member), 111

lv_gauge_get_critical_value (C++ function), 110

lv_gauge_get_label_count (C++ function), 110

lv_gauge_get_line_count (C++ function), 110

lv_gauge_get_max_value (C++ function), 110

lv_gauge_get_min_value (C++ function), 110

lv_gauge_get_needle_count (C++ function), 110

lv_gauge_get_scale_angle (C++ function), 110

lv_gauge_get_style (C++ function), 110

lv_gauge_get_value (C++ function), 109

lv_gauge_set_critical_value (C++ function), 109

lv_gauge_set_needle_count (C++ function), 108

lv_gauge_set_range (C++ function), 109

lv_gauge_set_scale (C++ function), 109

lv_gauge_set_style (C++ function), 109

lv_gauge_set_value (C++ function), 109

LV_GAUGE_STYLE_MAIN (C++ enumerator), 108

lv_gauge_style_t (C++ type), 108

lv_img_create (C++ function), 115

LV_IMG_DECLARE (C macro), 114

lv_img_ext_t (C++ class), 116

lv_img_ext_t::auto_size (C++ member), 116

lv_img_ext_t::cf (C++ member), 116

lv_img_ext_t::h (C++ member), 116

lv_img_ext_t::offset (C++ member), 116

lv_img_ext_t::src (C++ member), 116

lv_img_ext_t::src_type (C++ member), 116

lv_img_ext_t::w (C++ member), 116

lv_img_get_auto_size (C++ function), 116

lv_img_get_file_name (C++ function), 115

lv_img_get_offset_x (C++ function), 116

lv_img_get_offset_y (C++ function), 116

lv_img_get_src (C++ function), 115

lv_img_get_style (C++ function), 116

lv_img_set_auto_size (C++ function), 115

lv_img_set_offset_x (C++ function), 115

lv_img_set_offset_y (C++ function), 115

lv_img_set_src (C++ function), 115

lv_img_set_style (C++ function), 115

LV_IMG_STYLE_MAIN (C++ enumerator), 114

lv_img_style_t (C++ type), 114

lv_imgbtn_create (C++ function), 119

lv_imgbtn_ext_t (C++ class), 121

lv_imgbtn_ext_t::act_cf (C++ member), 121

lv_imgbtn_ext_t::btn (C++ member), 121

lv_imgbtn_ext_t::img_src (C++ member), 121

lv_imgbtn_get_src (C++ function), 120

lv_imgbtn_get_state (C++ function), 120

lv_imgbtn_get_style (C++ function), 121

lv_imgbtn_get_toggle (C++ function), 121

lv_imgbtn_set_src (C++ function), 120

lv_imgbtn_set_state (C++ function), 120

lv_imgbtn_set_style (C++ function), 120

lv_imgbtn_set_toggle (C++ function), 120

LV_IMGBTN_STYLE_INA (C++ enumerator), 119

LV_IMGBTN_STYLE_PR (C++ enumerator), 119

LV_IMGBTN_STYLE_REL (C++ enumerator), 119

lv_imgbtn_style_t (C++ type), 119

LV_IMGBTN_STYLE_TGL_PR (C++ enumerator), 119

[LV_IMGBTN_STYLE_TGL_REL \(C++ *enumerator*\), 119](#)
[lv_imgbtn_toggle \(C++ *function*\), 120](#)
[lv_kb_create \(C++ *function*\), 124](#)
[lv_kb_def_event_cb \(C++ *function*\), 126](#)
[lv_kb_ext_t \(C++ *class*\), 126](#)
[lv_kb_ext_t::btnm \(C++ *member*\), 126](#)
[lv_kb_ext_t::cursor_mng \(C++ *member*\), 126](#)
[lv_kb_ext_t::mode \(C++ *member*\), 126](#)
[lv_kb_ext_t::ta \(C++ *member*\), 126](#)
[lv_kb_get_cursor_manage \(C++ *function*\), 126](#)
[lv_kb_get_map_array \(C++ *function*\), 126](#)
[lv_kb_get_mode \(C++ *function*\), 126](#)
[lv_kb_get_style \(C++ *function*\), 126](#)
[lv_kb_get_ta \(C++ *function*\), 125](#)
[LV_KB_MODE_NUM \(C++ *enumerator*\), 124](#)
[lv_kb_mode_t \(C++ *type*\), 124](#)
[LV_KB_MODE_TEXT \(C++ *enumerator*\), 124](#)
[lv_kb_set_ctrl_map \(C++ *function*\), 125](#)
[lv_kb_set_cursor_manage \(C++ *function*\), 125](#)
[lv_kb_set_map \(C++ *function*\), 125](#)
[lv_kb_set_mode \(C++ *function*\), 125](#)
[lv_kb_set_style \(C++ *function*\), 125](#)
[lv_kb_set_ta \(C++ *function*\), 124](#)
[LV_KB_STYLE_BG \(C++ *enumerator*\), 124](#)
[LV_KB_STYLE_BTN_INA \(C++ *enumerator*\), 124](#)
[LV_KB_STYLE_BTN_PR \(C++ *enumerator*\), 124](#)
[LV_KB_STYLE_BTN_REL \(C++ *enumerator*\), 124](#)
[LV_KB_STYLE_BTN_TGL_PR \(C++ *enumerator*\), 124](#)
[LV_KB_STYLE_BTN_TGL_REL \(C++ *enumerator*\), 124](#)
[lv_kb_style_t \(C++ *type*\), 124](#)
[LV_LAYOUT_CENTER \(C++ *enumerator*\), 97](#)
[LV_LAYOUT_COL_L \(C++ *enumerator*\), 97](#)
[LV_LAYOUT_COL_M \(C++ *enumerator*\), 97](#)
[LV_LAYOUT_COL_R \(C++ *enumerator*\), 97](#)
[LV_LAYOUT_GRID \(C++ *enumerator*\), 97](#)
[LV_LAYOUT_OFF \(C++ *enumerator*\), 97](#)
[LV_LAYOUT_PRETTY \(C++ *enumerator*\), 97](#)
[LV_LAYOUT_ROW_B \(C++ *enumerator*\), 97](#)
[LV_LAYOUT_ROW_M \(C++ *enumerator*\), 97](#)
[LV_LAYOUT_ROW_T \(C++ *enumerator*\), 97](#)
[lv_layout_t \(C++ *type*\), 96](#)