
LVGL Documentation v7.6.1-dev

Contributors of LVGL

Oct 02, 2020

CONTENTS

1	Introduction	2
1.1	Key features	2
1.2	Requirements	3
1.3	License	3
1.4	Repository layout	3
1.5	Release policy	4
1.6	FAQ	5
2	Get started	8
2.1	Quick overview	8
2.2	Simulator on PC	17
2.3	STM32	19
2.4	NXP	20
2.5	Espressif (ESP32)	20
2.6	Arduino	20
2.7	Micropython	21
2.8	NuttX RTOS	24
3	Porting	27
3.1	System overview	27
3.2	Set-up a project	28
3.3	Display interface	29
3.4	Input device interface	32
3.5	Tick interface	36
3.6	Task Handler	36
3.7	Sleep management	36
3.8	Operating system and interrupts	37
3.9	Logging	38
4	Overview	40
4.1	Objects	40
4.2	Layers	45
4.3	Events	47
4.4	Styles	50
4.5	Input devices	76
4.6	Displays	78
4.7	Fonts	83
4.8	Images	90
4.9	File system	98
4.10	Animations	100

4.11	Tasks	102
4.12	Drawing	104
5	Widgets	110
5.1	Base object (lv_obj)	110
5.2	Arc (lv_arc)	116
5.3	Bar (lv_bar)	119
5.4	Button (lv_btn)	121
5.5	Button matrix (lv_btnmatrix)	125
5.6	Calendar (lv_calendar)	128
5.7	Canvas (lv_canvas)	132
5.8	Checkbox (lv_cb)	136
5.9	Chart (lv_chart)	139
5.10	Container (lv_cont)	144
5.11	color picker (lv_cpicker)	147
5.12	Drop-down list (lv_dropdown)	149
5.13	Gauge (lv_gauge)	153
5.14	Image (lv_img)	156
5.15	Image button (lv_imgbtn)	161
5.16	Keyboard (lv_keyboard)	163
5.17	Label (lv_label)	165
5.18	LED (lv_led)	170
5.19	Line (lv_line)	172
5.20	List (lv_list)	174
5.21	Line meter (lv_lmeter)	177
5.22	Message box (lv_msgbox)	179
5.23	Object mask (lv_objmask)	183
5.24	Page (lv_page)	187
5.25	Roller (lv_roller)	190
5.26	Slider (lv_slider)	193
5.27	Spinbox (lv_spinbox)	196
5.28	Spinner (lv_spinner)	198
5.29	Switch (lv_switch)	200
5.30	Table (lv_table)	202
5.31	Tabview (lv_tabview)	205
5.32	Text area (lv_textarea)	208
5.33	Tile view (lv_tileview)	214
5.34	Window (lv_win)	217
6	Contributing	220
6.1	Introduction	220
6.2	Pull request	221
6.3	Developer Certification of Origin (DCO)	222
6.4	When you get started with LVGL	223
6.5	When you already use LVGL	224
6.6	When you are confident with LVGL	226

PDF version: LVGL.pdf

INTRODUCTION

LVGL (Light and Versatile Graphics Library) 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 such as buttons, charts, lists, sliders, images etc.
- Advanced graphics with animations, anti-aliasing, opacity, smooth scrolling
- Various input devices such as touchpad, mouse, keyboard, encoder etc.
- Multi-language support with UTF-8 encoding
- Multi-display support, i.e. use more TFT, monochrome displays simultaneously
- Fully customizable graphic elements
- Hardware independent to use with any microcontroller or display
- Scalable to operate with little memory (64 kB Flash, 16 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 a PC without embedded hardware
- Binding to MicroPython
- Tutorials, examples, themes for rapid GUI design
- Documentation is available as online and offline
- Free and open-source under MIT license

1.2 Requirements

Basically, every modern controller (which is able to drive a display) is suitable to run LVGL. The minimal requirements are:

1.3 License

The LVGL project (including all repositories) is licensed under [MIT license](#). It means you can use it even in commercial projects.

It's not mandatory but we highly appreciate it if you write a few words about your project in the [My projects](#) category of the Forum or a private message from [lvgl.io](#).

Although you can get LVGL for free there is a huge work behind it. It's created by a group of volunteers who made it available for you in their free time.

To make the LVGL project sustainable, please consider [Contributing](#) to the project. You can choose from *many ways of contributions* such as simply writing a tweet about you are using LVGL, fixing bugs, translating the documentation, or even becoming a maintainer.

1.4 Repository layout

All repositories of the LVGL project are hosted on GitHub: <https://github.com/lvgl>

You find these repositories there:

- [lvgl](#) The library itself
- [lv_examples](#) Examples and demos
- [lv_drivers](#) Display and input device drivers
- [docs](#) Source of the documentation's site (<https://docs.lvgl.io>)
- [blog](#) Source of the blog's site (<https://blog.lvgl.io>)
- [sim](#) Source of the online simulator's site (<https://sim.lvgl.io>)
- [lv_sim_...](#) Simulator projects for various IDEs and platforms
- [lv_port_...](#) LVGL ports to development boards
- [lv_binding_..](#) Bindings to other languages
- [lv_...](#) Ports to other platforms

The [lvgl](#), [lv_examples](#) and [lv_drivers](#) are the core repositories which get the most attention regarding maintenance.

1.5 Release policy

The core repositories follow the rules of [Semantic versioning](#):

- Major versions for incompatible API changes. E.g. v5.0.0, v6.0.0
- Minor version for new but backward-compatible functionalities. E.g. v6.1.0, v6.2.0
- Patch version for backward-compatible bug fixes. E.g. v6.1.1, v6.1.2

1.5.1 Branches

The core repositories have at least the following branches:

- **master** latest version, patches are merged directly here.
- **dev** merge new features here until they are merged into **master**.
- **release/vX** stable versions of the major releases

1.5.2 Release cycle

LVGL has 2 weeks release cycle. On every first and third Tuesday of a month:

1. A major, minor or bug fix release is created (based on the new features) from the **master** branch
2. **master** is merged into **release/vX**
3. Immediately after the release **dev** is merged into **master**
4. In the upcoming 2 weeks the new features in **master** can be tested
5. Bug fixes are merged directly into **master**
6. After 2 weeks start again from the first point

1.5.3 Tags

Tags like **vX.Y.Z** are created for every release.

1.5.4 Changelog

The changes are recorded in [CHANGELOG.md](#).

1.5.5 Side projects

The [docs](#) is rebuilt on every release. By default, the **latest** documentation is displayed which is for the current **master** branch of lvgl. The documentation of earlier versions is available from the menu on the left.

The simulator, porting, and other projects are updated with best effort. Pull requests are welcome if you updated one of them.

1.5.6 Version support

In the core repositories each major version has a branch (e.g. `release/v6`). All the minor and patch releases of that major version are merged there.

It makes possible to add fixed older versions without bothering the newer ones.

All major versions are officially supported for 1 year.

1.6 FAQ

1.6.1 Where can I ask questions?

You can ask questions in the Forum: <https://forum.lvgl.io/>.

We use [GitHub issues](#) for development related discussion. So you should use them only if your question or issue is tightly related to the development of the library.

1.6.2 Is my MCU/hardware supported?

Every MCU which is capable of driving a display via Parallel port, SPI, RGB interface or anything else and fulfills the *Requirements* is supported by LVGL.

It includes:

- "Common" MCUs like STM32F, STM32H, NXP Kinetis, LPC, iMX, dsPIC33, PIC32 etc.
- Bluetooth, GSM, WiFi modules like Nordic NRF and Espressif ESP32
- Linux frame buffer like `/dev/fb0` which includes Single-board computers too like Raspberry Pi
- And anything else with a strong enough MCU and a periphery to drive a display

1.6.3 Is my display supported?

LVGL needs just one simple driver function to copy an array of pixels into a given area of the display. If you can do this with your display then you can use that display with LVGL.

Some examples of the supported display types:

- TFTs with 16 or 24 bit color depth
- Monitors with HDMI port
- Small monochrome displays
- Gray-scale displays
- even LED matrices
- or any other display where you can control the color/state of the pixels

See the *Porting* section to learn more.

1.6.4 Nothing happens, my display driver is not called. What have I missed?

Be sure you are calling `lv_tick_inc(x)` in an interrupt and `lv_task_handler()` in your main `while(1)`.

Learn more in the *Tick* and *Task handler* section.

1.6.5 Why the display driver is called only once? Only the upper part of the display is refreshed.

Be sure you are calling `lv_disp_flush_ready(drv)` at the end of your "*display flush callback*".

1.6.6 Why I see only garbage on the screen?

Probably there a bug in your display driver. Try the following code without using LVGL. You should see a square with red-blue gradient

```
#define BUF_W 20
#define BUF_H 10

lv_color_t buf[BUF_W * BUF_H];
lv_color_t * buf_p = buf;
uint16_t x, y;
for(y = 0; y < BUF_H; y++) {
    lv_color_t c = lv_color_mix(LV_COLOR_BLUE, LV_COLOR_RED, (y * 255) / BUF_H);
    for(x = 0; x < BUF_W; x++){
        (*buf_p) = c;
        buf_p++;
    }
}

lv_area_t a;
a.x1 = 10;
a.y1 = 40;
a.x2 = a.x1 + BUF_W - 1;
a.y2 = a.y1 + BUF_H - 1;
my_flush_cb(NULL, &a, buf);
```

1.6.7 Why I see non-sense colors on the screen?

Probably LVGL's color format is not compatible with your displays color format. Check `LV_COLOR_DEPTH` in `lv_conf.h`.

If you are using 16 bit colors with SPI (or other byte-oriented interface) probably you need to set `LV_COLOR_16_SWAP 1` in `lv_conf.h`. It swaps the upper and lower bytes of the pixels.

1.6.8 How to speed up my UI?

- Turn on compiler optimization and enable cache if your MCU has
- Increase the size of the display buffer
- Use 2 display buffers and flush the buffer with DMA (or similar periphery) in the background
- Increase the clock speed of the SPI or Parallel port if you use them to drive the display
- If your display has SPI port consider changing to a model with parallel because it has much higher throughput
- Keep the display buffer in the internal RAM (not in external SRAM) because LVGL uses it a lot and it should have a small access time

1.6.9 How to reduce flash/ROM usage?

You can disable all the unused features (such as animations, file system, GPU etc.) and object types in *lv_conf.h*.

If you are using GCC you can add

- `-fdata-sections -ffunction-sections` compiler flags
- `--gc-sections` linker flag

to remove unused functions and variables from the final binary

1.6.10 How to reduce the RAM usage

- Lower the size of the *Display buffer*
- Reduce `LV_MEM_SIZE` in *lv_conf.h*. This memory used when you create objects like buttons, labels, etc.
- To work with lower `LV_MEM_SIZE` you can create the objects only when required and deleted them when they are not required anymore

1.6.11 How to work with an operating system?

To work with an operating system where tasks can interrupt each other (preemptive) you should protect LVGL related function calls with a mutex. See the *Operating system and interrupts* section to learn more.

GET STARTED

There are several ways to get your feet wet with LVGL. This list shows the recommended way of learning the library:

1. Check the [Online demos](#) to see LVGL in action (3 minutes)
2. Read the [Introduction](#) page of the documentation (5 minutes)
3. Read the [Quick overview](#) page of the documentation (15 minutes)
4. Set up a [Simulator](#) (10 minutes)
5. Try out some [Examples](#)
6. Port LVGL to a board. See the [Porting](#) guide or check the ready to use [Projects](#)
7. Read the [Overview](#) page to get a better understanding of the library. (2-3 hours)
8. Check the documentation of the [Widgets](#) to see their features and usage
9. If you have questions got to the [Forum](#)
10. Read the [Contributing](#) guide to see how you can help to improve LVGL (15 minutes)

2.1 Quick overview

Here you can learn the most important things about LVGL. You should read it first to get a general impression and read the detailed [Porting](#) and [Overview](#) sections after that.

2.1.1 Get started in a simulator

Instead of porting LVGL to an embedded hardware, it's highly recommended to get started in a simulator first.

LVGL is ported to many IDEs to be sure you will find your favourite one. Go to [Simulators](#) to get ready-to-use projects which can be run on your PC. This way you can save the porting for now and make some experience with LVGL immediately.

2.1.2 Add LVGL into your project

The following steps show how to setup LVGL on an embedded system with a display and a touchpad.

- [Download](#) or Clone the library from GitHub with `git clone https://github.com/lvgl/lvgl.git`
- Copy the `lvgl` folder into your project
- Copy `lvgl/lv_conf_templ.h` as `lv_conf.h` next to the `lvgl` folder, change the first `#if 0` to `1` to enable the file's content and set at least `LV_HOR_RES_MAX`, `LV_VER_RES_MAX` and `LV_COLOR_DEPTH` defines.
- Include `lvgl/lvgl.h` where you need to use LVGL related functions.
- Call `lv_tick_inc(x)` every `x` milliseconds **in a Timer or Task** (`x` should be between 1 and 10). It is required for the internal timing of LVGL.
- Call `lv_init()`
- Create a display buffer for LVGL. LVGL will render the graphics here first, and send the rendered image to the display. The buffer size can be set freely but 1/10 screen size is a good starting point.

```
static lv_disp_buf_t disp_buf;
static lv_color_t buf[LV_HOR_RES_MAX * LV_VER_RES_MAX / 10];           /
↪/*Declare a buffer for 1/10 screen size*/
lv_disp_buf_init(&disp_buf, buf, NULL, LV_HOR_RES_MAX * LV_VER_RES_MAX / 10);  /
↪/*Initialize the display buffer*/
```

- Implement and register a function which can **copy the rendered image** to an area of your display:

```
lv_disp_drv_t disp_drv;           /*Descriptor of a display driver*/
lv_disp_drv_init(&disp_drv);       /*Basic initialization*/
disp_drv.flush_cb = my_disp_flush; /*Set your driver function*/
disp_drv.buffer = &disp_buf;       /*Assign the buffer to the display*/
lv_disp_drv_register(&disp_drv);    /*Finally register the driver*/

void my_disp_flush(lv_disp_drv_t * disp, const lv_area_t * area, lv_color_t * color_p)
{
    int32_t x, y;
    for(y = area->y1; y <= area->y2; y++) {
        for(x = area->x1; x <= area->x2; x++) {
            set_pixel(x, y, *color_p); /* Put a pixel to the display.*/
            color_p++;
        }
    }

    lv_disp_flush_ready(disp);      /* Indicate you are ready with the flushing*/
}
```

- Implement and register a function which can **read an input device**. E.g. for a touch pad:

```
lv_indev_drv_t indev_drv;           /*Descriptor of a input device driver*/
lv_indev_drv_init(&indev_drv);       /*Basic initialization*/
indev_drv.type = LV_INDEV_TYPE_POINTER; /*Touch pad is a pointer-like device*/
indev_drv.read_cb = my_touchpad_read; /*Set your driver function*/
lv_indev_drv_register(&indev_drv);    /*Finally register the driver*/

bool my_touchpad_read(lv_indev_t * indev, lv_indev_data_t * data)
{
    }
```

(continues on next page)

(continued from previous page)

```

data->state = touchpad_is_pressed() ? LV_INDEV_STATE_PR : LV_INDEV_STATE_REL;
if(data->state == LV_INDEV_STATE_PR) touchpad_get_xy(&data->point.x, &data->point.
↪y);

return false; /*Return `false` because we are not buffering and no more data to_
↪read*/
}

```

- Call `lv_task_handler()` periodically every few milliseconds in the main `while(1)` loop, in Timer interrupt or in an Operation system task. It will redraw the screen if required, handle input devices etc.

For a more detailed guide go to the [Porting](#) section.

2.1.3 Learn the basics

Widgets

The graphical elements like Buttons, Labels, Sliders, Charts etc are called objects or widgets in LVGL. Go to [Widgets](#) to see the full list of available widgets.

Every object has a parent object where it is create. For example if a label is created on a button, the button is the parent of label. The child object moves with the parent and if the parent is deleted the children will be deleted too.

Children can be visible only on their parent. In other words, the parts of the children out of the parent are clipped.

A *screen* is the "root" parent. You can have any number of screens. To get the current screen call `lv_scr_act()`, and to load a screen use `lv_scr_load(scr1)`.

You can create a new object with `lv_<type>_create(parent, obj_to_copy)`. It will return an `lv_obj_t *` variable which should be used as a reference to the object to set its parameters. The first parameter is the desired *parent*, the second parameters can be an object to copy (NULL if unused). For example:

```
lv_obj_t * slider1 = lv_slider_create(lv_scr_act(), NULL);
```

To set some basic attribute `lv_obj_set_<parameters_name>(obj, <value>)` function can be used. For example:

```
lv_obj_set_x(btn1, 30);
lv_obj_set_y(btn1, 10);
lv_obj_set_size(btn1, 200, 50);
```

The objects has type specific parameters too which can be set by `lv_<type>_set_<parameters_name>(obj, <value>)` functions. For example:

```
lv_slider_set_value(slider1, 70, LV_ANIM_ON);
```

To see the full API visit the documentation of the widgets or the related header file (e.g. `lvgl/src/lv_widgets/lv_slider.h`).

Events

Events are used to inform the user if something has happened with an object. You can assign a callback to an object which will be called if the object is clicked, released, dragged, being deleted etc. It should look like this:

```
lv_obj_set_event_cb(btn, btn_event_cb);           /*Assign a callback to the_
↪button*/

...

void btn_event_cb(lv_obj_t * btn, lv_event_t event)
{
    if(event == LV_EVENT_CLICKED) {
        printf("Clicked\n");
    }
}
```

Learn more about the events in the [Event overview](#) section.

Parts

Widgets might be built from one or more parts. For example a button has only one part called `LV_BTN_PART_MAIN`. However, a *Page* has `LV_PAGE_PART_BG`, `LV_PAGE_PART_SCROLLABLE`, `LV_PAGE_PART_SCROLLBAR` and `LV_PAGE_PART_EDGE_FLAG`.

Some parts are *virtual* (they are not real object, just drawn on the fly, such as the scrollbar of a page) but other parts are *real* (they are real object, such as the scrollable part of the page).

Parts come into play when you want to set the styles and states of a given part of an object. (See below)

States

The objects can be in a combination of the following states:

- **LV_STATE_DEFAULT** Normal, released
- **LV_STATE_CHECKED** Toggled or checked
- **LV_STATE_FOCUSED** Focused via keypad or encoder or clicked via touchpad/mouse
- **LV_STATE_EDITED** Edit by an encoder
- **LV_STATE_HOVERED** Hovered by mouse (not supported now)
- **LV_STATE_PRESSED** Pressed
- **LV_STATE_DISABLED** Disabled or inactive

For example if you press an object is automatically get the `LV_STATE_PRESSED` state and when you release is it will be removed.

To get the current state use `lv_obj_get_state(obj, part)`. It will return the **ORed** states. For example it's a valid state for a checkbox: `LV_STATE_CHECKED | LV_STATE_PRESSED | LV_STATE_FOCUSED`

Styles

Styles can be assigned to the parts objects to changed their appearance. A style can describe for example the background color, border width, text font and so on. See the full list [here](#).

The styles can be cascaded (similarly to CSS). It means you can add more styles to a part of an object. For example `style_btn` can set a default button appearance, and `style_btn_red` can overwrite some properties to make the button red-

Every style property you set is specific to a state. For example is you can set different background color for `LV_STATE_DEFAULT` and `LV_STATE_PRESSED`. The library finds the best match between the state of the given part and the available style properties. For example if the object is in pressed state and the border width is specified for pressed state, then it will be used. However, if it's not specified for pressed state, the `LV_STATE_DEFAULT`'s border width will be used. If the border width not defined for `LV_STATE_DEFAULT` either, a default value will be used.

Some properties (typically the text-related ones) can be inherited. It means if a property is not set in an object it will be searched in its parents too. For example you can set the font once in the screen's style and every text will inherit it by default.

Local style properties also can be added to the objects.

Themes

Themes are the default styles of the objects. The styles from the themes are applied automatically when the objects are created.

You can select the theme to use in `lv_conf.h`.

2.1.4 Examples

Button with label



```

#include "../../lv_examples.h"

static void btn_event_cb(lv_obj_t * btn, lv_event_t event)
{
    if(event == LV_EVENT_CLICKED) {
        static uint8_t cnt = 0;
        cnt++;

        /*Get the first child of the button which is the label and change its text*/
        lv_obj_t * label = lv_obj_get_child(btn, NULL);
        lv_label_set_text_fmt(label, "Button: %d", cnt);
    }
}

/**
 * Create a button with a label and react on Click event.
 */
void lv_ex_get_started_1(void)
{
    lv_obj_t * btn = lv_btn_create(lv_scr_act(), NULL); /*Add a button the
↪current screen*/
    lv_obj_set_pos(btn, 10, 10); /*Set its position*/
    lv_obj_set_size(btn, 120, 50); /*Set its size*/
    lv_obj_set_event_cb(btn, btn_event_cb); /*Assign a callback to
↪the button*/

    lv_obj_t * label = lv_label_create(btn, NULL); /*Add a label to the
↪button*/
    lv_label_set_text(label, "Button"); /*Set the labels text*/
}

```

Styling buttons




```

#include "../../lv_examples.h"

/**
 * Create styles from scratch for buttons.
 */
void lv_ex_get_started_2(void)
{
    static lv_style_t style_btn;
    static lv_style_t style_btn_red;

    /*Create a simple button style*/
    lv_style_init(&style_btn);
    lv_style_set_radius(&style_btn, LV_STATE_DEFAULT, 10);
    lv_style_set_bg_opa(&style_btn, LV_STATE_DEFAULT, LV_OPA_COVER);
    lv_style_set_bg_color(&style_btn, LV_STATE_DEFAULT, LV_COLOR_SILVER);
    lv_style_set_bg_grad_color(&style_btn, LV_STATE_DEFAULT, LV_COLOR_GRAY);
    lv_style_set_bg_grad_dir(&style_btn, LV_STATE_DEFAULT, LV_GRAD_DIR_VER);

    /*Swap the colors in pressed state*/
    lv_style_set_bg_color(&style_btn, LV_STATE_PRESSED, LV_COLOR_GRAY);
    lv_style_set_bg_grad_color(&style_btn, LV_STATE_PRESSED, LV_COLOR_SILVER);

    /*Add a border*/
    lv_style_set_border_color(&style_btn, LV_STATE_DEFAULT, LV_COLOR_WHITE);
    lv_style_set_border_opa(&style_btn, LV_STATE_DEFAULT, LV_OPA_70);
    lv_style_set_border_width(&style_btn, LV_STATE_DEFAULT, 2);

    /*Different border color in focused state*/
    lv_style_set_border_color(&style_btn, LV_STATE_FOCUSED, LV_COLOR_BLUE);
    lv_style_set_border_color(&style_btn, LV_STATE_FOCUSED | LV_STATE_PRESSED, LV_
    ↪COLOR_NAVY);

    /*Set the text style*/
    lv_style_set_text_color(&style_btn, LV_STATE_DEFAULT, LV_COLOR_WHITE);

    /*Make the button smaller when pressed*/
    lv_style_set_transform_height(&style_btn, LV_STATE_PRESSED, -5);
    lv_style_set_transform_width(&style_btn, LV_STATE_PRESSED, -10);
    #if LV_USE_ANIMATION
    /*Add a transition to the size change*/
    static lv_anim_path_t path;
    lv_anim_path_init(&path);
    lv_anim_path_set_cb(&path, lv_anim_path_overshoot);

    lv_style_set_transition_prop_1(&style_btn, LV_STATE_DEFAULT, LV_STYLE_TRANSFORM_
    ↪HEIGHT);
    lv_style_set_transition_prop_2(&style_btn, LV_STATE_DEFAULT, LV_STYLE_TRANSFORM_
    ↪WIDTH);
    lv_style_set_transition_time(&style_btn, LV_STATE_DEFAULT, 300);
    lv_style_set_transition_path(&style_btn, LV_STATE_DEFAULT, &path);
    #endif

    /*Create a red style. Change only some colors.*/
    lv_style_init(&style_btn_red);
    lv_style_set_bg_color(&style_btn_red, LV_STATE_DEFAULT, LV_COLOR_RED);
    lv_style_set_bg_grad_color(&style_btn_red, LV_STATE_DEFAULT, LV_COLOR_MAROON);

```

(continues on next page)

(continued from previous page)

```

lv_style_set_bg_color(&style_btn_red, LV_STATE_PRESSED, LV_COLOR_MAROON);
lv_style_set_bg_grad_color(&style_btn_red, LV_STATE_PRESSED, LV_COLOR_RED);
lv_style_set_text_color(&style_btn_red, LV_STATE_DEFAULT, LV_COLOR_WHITE);
#if LV_USE_BTN
    /*Create buttons and use the new styles*/
    lv_obj_t * btn = lv_btn_create(lv_scr_act(), NULL);           /*Add a button the
↪current screen*/
    lv_obj_set_pos(btn, 10, 10);                                  /*Set its position*/
    lv_obj_set_size(btn, 120, 50);                                /*Set its size*/
    lv_obj_reset_style_list(btn, LV_BTN_PART_MAIN);               /*Remove the styles
↪coming from the theme*/
    lv_obj_add_style(btn, LV_BTN_PART_MAIN, &style_btn);

    lv_obj_t * label = lv_label_create(btn, NULL);                /*Add a label to the
↪button*/
    lv_label_set_text(label, "Button");                            /*Set the labels text*/

    /*Create a new button*/
    lv_obj_t * btn2 = lv_btn_create(lv_scr_act(), btn);
    lv_obj_set_pos(btn2, 10, 80);
    lv_obj_set_size(btn2, 120, 50);                               /*Set its size*/
    lv_obj_reset_style_list(btn2, LV_BTN_PART_MAIN);              /*Remove the styles
↪coming from the theme*/
    lv_obj_add_style(btn2, LV_BTN_PART_MAIN, &style_btn);
    lv_obj_add_style(btn2, LV_BTN_PART_MAIN, &style_btn_red);     /*Add the red style
↪on top of the current */
    lv_obj_set_style_local_radius(btn2, LV_BTN_PART_MAIN, LV_STATE_DEFAULT, LV_RADIUS_
↪CIRCLE); /*Add a local style*/

    label = lv_label_create(btn2, NULL);                          /*Add a label to the button*/
    lv_label_set_text(label, "Button 2");                          /*Set the labels text*/
#endif
}

```

Slider and alignment



```
#include "../lv_examples.h"

static lv_obj_t * label;

static void slider_event_cb(lv_obj_t * slider, lv_event_t event)
{
    if(event == LV_EVENT_VALUE_CHANGED) {
        /*Refresh the text*/
        lv_label_set_text_fmt(label, "%d", lv_slider_get_value(slider));
    }
}

/**
 * Create a slider and write its value on a label.
 */
void lv_ex_get_started_3(void)
{
    /* Create a slider in the center of the display */
    lv_obj_t * slider = lv_slider_create(lv_scr_act(), NULL);
    lv_obj_set_width(slider, 200); /*Set the width*/
    lv_obj_align(slider, NULL, LV_ALIGN_CENTER, 0, 0); /*Align to the center of
↳the parent (screen)*/
    lv_obj_set_event_cb(slider, slider_event_cb); /*Assign an event function*/

    /* Create a label below the slider */
    label = lv_label_create(lv_scr_act(), NULL);
    lv_label_set_text(label, "0");
    lv_obj_set_auto_realign(slider, true); /*To keep center
↳alignment when the width of the text changes*/
    lv_obj_align(label, slider, LV_ALIGN_OUT_BOTTOM_MID, 0, 20); /*Align below the
↳slider*/
}
```

(continues on next page)

(continued from previous page)

2.1.5 Micropython

Learn more about *Micropython*.

```
# Create a Button and a Label
scr = lv.obj()
btn = lv.btn(scr)
btn.align(lv.scr_act(), lv.ALIGN.CENTER, 0, 0)
label = lv.label(btn)
label.set_text("Button")

# Load the screen
lv.scr_load(scr)
```

2.2 Simulator on PC

You can try out the LVGL **using only your PC** (i.e. without any development boards). The LVGL will run on a simulator environment on the PC where anyone can write and experiment the real LVGL applications.

Simulator on the PC have the following advantages:

- Hardware independent - Write a code, run it on the PC and see the result on the PC monitor.
- Cross-platform - Any Windows, Linux or OSX PC can run the PC simulator.
- Portability - the written code is portable, which means you can simply copy it when using an embedded hardware.
- Easy Validation - The simulator is also very useful to report bugs because it means common platform for every user. So it's a good idea to reproduce a bug in simulator and use the code snippet in the [Forum](#).

2.2.1 Select an IDE

The simulator is ported to various IDEs (Integrated Development Environments). Choose your favorite IDE, read its README on GitHub, download the project, and load it to the IDE.

You can use any IDEs for the development but, for simplicity, the configuration for Eclipse CDT is focused in this tutorial. The following section describes the set-up guide of Eclipse CDT in more details.

Note: If you are on Windows, it's usually better to use the Visual Studio or CodeBlocks projects instead. They work out of the box without requiring extra steps.

2.2.2 Set-up Eclipse CDT

Install Eclipse CDT

Eclipse CDT is a C/C++ IDE.

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`

Note: If you are using other distros, then please refer and install 'Java Runtime Environment' suitable to your distro. Note: If you are using macOS and get a "Failed to create the Java Virtual Machine" error, uninstall any other Java JDK installs and install Java JDK 8u. This should fix the problem.

You can download Eclipse's CDT from: <https://www.eclipse.org/cdt/downloads.php>. 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 installing MinGW, 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. Decompress 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 are using **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, then please refer [this tutorial](#) to get started with SDL.

Pre-configured project

A pre-configured graphics library project (based on the latest release) is always available to get started easily. You can find the latest one on [GitHub](#). (Please note that, 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 the path, check that path and copy (and unzip) the downloaded pre-configured project there. After that, you can accept the workspace path. Of course you can modify this path but, in that case copy the project to the corresponding 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 LVGL 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, then 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 LVGL in the practice or begin the development on your PC.

2.3 STM32

TODO

2.4 NXP

TODO

2.5 Espressif (ESP32)

2.6 Arduino

The [core LVGL library](#) and the [examples](#) are directly available as Arduino libraries.

Note that you need to choose a powerful enough board to run LVGL and your GUI. See the [requirements of LVGL](#).

For example ESP32 is a good candidate to create your UI with LVGL.

2.6.1 Get the LVGL Arduino library

LVGL can be installed via Arduino IDE Library Manager or as an .ZIP library. It will also install [lv_exmaples](#) which contains a lot of examples and demos to try LVGL.

2.6.2 Set up drivers

To get started it's recommended to use [TFT_eSPI](#) library as a TFT driver to simplify testing. To make it work setup [TFT_eSPI](#) according to your TFT display type via editing either

- [User_Setup.h](#)
- or by selecting a configuration in the [User_Setup_Select.h](#)

Both files are located in [TFT_eSPI](#) library's folder.

2.6.3 Configure LVGL

LVGL has its own configuration file called [lv_conf.h](#). When LVGL is installed the followings needs to be done to configure it:

1. Go to directory of the installed Arduino libraries
2. Go to [lvgl](#) and copy [lv_conf_template.h](#) as [lv_conf.h](#) next to the [lvgl](#) folder.
3. Open [lv_conf.h](#) and change the first `#if 0` to `#if 1`
4. Set the resolution of your display in [LV_HOR_RES_MAX](#) and [LV_VER_RES_MAX](#)
5. Set the color depth of you display in [LV_COLOR_DEPTH](#)
6. Set [LV_TICK_CUSTOM](#) 1

2.6.4 Configure the examples

`lv_examples` can be configured similarly to LVGL but its configuration file is called `lv_ex_conf.h`.

1. Go to directory of the installed Arduino libraries
2. Go to `lv_examples` and copy `lv_ex_template.h` as `lv_ex_conf.h` next to the `lv_examples` folder.
3. Open `lv_ex_conf.h` and change the first `#if 0` to `#if 1`
4. Enable the demos you want to use. (The small examples starting with `lv_ex_...()` are always enabled.)

2.6.5 Initialize LVGL and run an example

Take a look at `LVGL_Arduino.ino` to see how to initialize LVGL. It also uses `TFT_eSPI` as driver.

In the INO file you can see how to register a display and a touch pad for LVGL and call an example.

Note that, there is no dedicated INO file for every example but you can call functions like `lv_ex_btn1()` or `lv_ex_slider1()` to run an example. For the full list of examples see the [README of lv_examples](#).

2.6.6 Debugging and logging

In case of trouble there are debug information inside LVGL. In the `LVGL_Arduino.ino` example there is `my_print` method, which allows to send this debug information to the serial interface. To enable this feature you have to edit `lv_conf.h` file and enable logging in section **log settings**:

```
/*Log settings*/
#define USE_LV_LOG      1  /*Enable/disable the log module*/
#if LV_USE_LOG
/* How important log should be added:
 * LV_LOG_LEVEL_TRACE      A lot of logs to give detailed information
 * LV_LOG_LEVEL_INFO       Log important events
 * LV_LOG_LEVEL_WARN        Log if something unwanted happened but didn't cause a
↳ problem
 * LV_LOG_LEVEL_ERROR       Only critical issue, when the system may fail
 * LV_LOG_LEVEL_NONE        Do not log anything
 */
# define LV_LOG_LEVEL      LV_LOG_LEVEL_WARN
```

After enabling log module and setting `LV_LOG_LEVEL` accordingly the output log is sent to the **Serial** port @ 115200 Baud rate.

2.7 Micropython

2.7.1 What is Micropython?

[Micropython](#) is Python for microcontrollers. Using Micropython, you can write Python3 code and run it even on a bare metal architecture with limited resources.

Highlights of Micropython

- **Compact** - Fits and runs within just 256k of code space and 16k of RAM. No OS is needed, although you can also run it with an OS, if you want.
 - **Compatible** - Strives to be as compatible as possible with normal Python (known as CPython).
 - **Versatile** - Supports many architectures (x86, x86-64, ARM, ARM Thumb, Xtensa).
 - **Interactive** - No need for the compile-flash-boot cycle. With the REPL (interactive prompt) you can type commands and execute them immediately, run scripts etc.
 - **Popular** - Many platforms are supported. The user base is growing bigger. Notable forks: [MicroPython](#), [CircuitPython](#), [MicroPython_ESP32_psRAM_LoBo](#)
 - **Embedded Oriented** - Comes with modules specifically for embedded systems, such as the [machine module](#) for accessing low-level hardware (I/O pins, ADC, UART, SPI, I2C, RTC, Timers etc.)
-

2.7.2 Why Micropython + LVGL?

Currently, Micropython *does not* have a good high-level GUI library by default. LVGL is an [Object Oriented Component Based](#) high-level GUI library, which seems to be a natural candidate to map into a higher level language, such as Python. LVGL is implemented in C and its APIs are in C.

Here are some advantages of using LVGL in Micropython:

- Develop GUI in Python, a very popular high level language. Use paradigms such as Object Oriented Programming.
- Usually, GUI development requires multiple iterations to get things right. With C, each iteration consists of **Change code > Build > Flash > Run**. In Micropython it's just **Change code > Run** ! You can even run commands interactively using the [REPL](#) (the interactive prompt)

Micropython + LVGL could be used for:

- Fast prototyping GUI.
 - Shorten the cycle of changing and fine-tuning the GUI.
 - Model the GUI in a more abstract way by defining reusable composite objects, taking advantage of Python's language features such as Inheritance, Closures, List Comprehension, Generators, Exception Handling, Arbitrary Precision Integers and others.
 - Make LVGL accessible to a larger audience. No need to know C in order to create a nice GUI on an embedded system. This goes well with [CircuitPython vision](#). CircuitPython was designed with education in mind, to make it easier for new or unexperienced users to get started with embedded development.
 - Creating tools to work with LVGL at a higher level (e.g. drag-and-drop designer).
-

2.7.3 So what does it look like?

TL;DR: It's very much like the C API, but Object Oriented for LVGL components.

Let's dive right into an example!

A simple example

```
import lvgl as lv
lv.init()
scr = lv.obj()
btn = lv.btn(scr)
btn.align(lv.scr_act(), lv.ALIGN.CENTER, 0, 0)
label = lv.label(btn)
label.set_text("Button")
lv.scr_load(scr)
```

2.7.4 How can I use it?

Online Simulator

If you want to experiment with LVGL + Micropython without downloading anything - you can use our online simulator! It's a fully functional LVGL + Micropython that runs entirely in the browser and allows you to edit a python script and run it.

[Click here to experiment on the online simulator](#)

Hello World

Note: the online simulator is available for lvgl v6 and v7.

PC Simulator

Micropython is ported to many platforms. One notable port is "unix", which allows you to build and run Micropython (+LVGL) on a Linux machine. (On a Windows machine you might need Virtual Box or WSL or MinGW or Cygwin etc.)

[Click here to know more information about building and running the unix port](#)

Embedded platform

At the end, the goal is to run it all on an embedded platform. Both Micropython and LVGL can be used on many embedded architectures, such as stm32, ESP32 etc. You would also need display and input drivers. We have some sample drivers (ESP32+ILI9341, as well as some other examples), but most chances are you would want to create your own input/display drivers for your specific purposes. Drivers can be implemented either in C as Micropython module, or in pure Micropython!

2.7.5 Where can I find more information?

- On the [Blog Post](#)
- On [lv_micropython README](#)
- On [lv_binding_micropython README](#)
- On LVGL forum (Feel free to ask anything!)
- On Micropython [docs](#) and [forum](#)

2.8 NuttX RTOS

2.8.1 What is NuttX?

[NuttX](#) is a mature and secure real-time operating system (RTOS) with an emphasis on technical standards compliance and small size. It is scalable from 8-bit to 64-bit microcontroller and microprocessors. Complaint with the Portable Operating System Interface (POSIX) and the American National Standards Institute (ANSI) standards and with many Linux-like subsystems. The best way to think about NuttX is thinking about a small Unix/Linux for microcontrollers.

Highlights of NuttX

- **Small** - Fits and runs within small microcontroller as small was 32KB Flash and 8KB of RAM.
- **Compliant** - Strives to be as compatible as possible with POSIX and Linux.
- **Versatile** - Supports many architectures (ARM, ARM Thumb, AVR, MIPS, OpenRISC, RISC-V 32-bit and 64-bit, RX65N, x86-64, Xtensa, Z80/Z180, etc).
- **Modular** - Its modular design allow developers to select only what really matters and use modules to include new features.
- **Popular** - NuttX is used by many companies around the world. Probably you already used a product with NuttX without knowing it was running NuttX.
- **Predictable** - NuttX is a preemptible Realtime kernel, then you can use it to create predictable applications for realtime control.

2.8.2 Why NuttX + LVGL?

Although NuttX has its own graphic library called [NX](#), LVGL is a good alternative because users could find more eyes-candy demos and reuse it from previous projects. LVGL is an [Object Oriented Component Based](#) high-level GUI library, that could fit very well for a RTOS with advanced features like NuttX. LVGL is implemented in C and its APIs are in C.

Here are some advantages of using LVGL in NuttX

- Develop GUI in Linux first and when it is done just compile it for NuttX, nothing more, no wasting of time.
- Usually, GUI development for low level RTOS requires multiple iterations to get things right. Where each iteration consists of **Change code** > **Build** > **Flash** > **Run**. Using LVGL, Linux and NuttX you can reduce this process and just test everything on your computer and when it is done, compile it on NuttX and that is it.

NuttX + LVGL could be used for

- GUI demos to demonstrate your board graphics capacities.
- Fast prototyping GUI for MVP (Minimum Viable Product) presentation.
- Easy way to visualize sensors data directly on the board without using a computer.
- Final products GUI without touchscreen (i.e. 3D Printer Interface using Rotary Encoder to Input data).
- Final products interface with touchscreen (and bells and whistles).

2.8.3 How to get started with NuttX and LVGL?

There are many boards in the NuttX mainline (<https://github.com/apache/incubator-nuttX>) with support for LVGL. Let's to use the **STM32F429IDISCOVERY** as example because it is a very popular board.

First you need to install the pre-requisite on your system

Let's to use Linux and example, for [Windows](#)

```
$ sudo apt-get install automake bison build-essential flex gcc-arm-none-eabi gperf
↪git libncurses5-dev libtool libusb-dev libusb-1.0.0-dev pkg-config kconfig-
↪frontends openocd
```

Now let's to create a workspace to save our files

```
$ mkdir ~/nuttxspace
$ cd ~/nuttxspace
```

Clone the NuttX and Apps repositories:

```
$ git clone https://github.com/apache/incubator-nuttX nuttx
$ git clone https://github.com/apache/incubator-nuttX-apps apps
```

Configure NuttX to use the stm32f429i-disco board and the LVGL Demo

```
$ ./tools/configure.sh stm32f429i-disco:lvgl
$ make
```

If everything went fine you should have now the file `nutttx.bin` to flash on your board:

```
$ ls -l nuttx.bin
-rwxrwxr-x 1 alan alan 287144 Jun 27 09:26 nuttx.bin
```

Flashing the firmware in the board using OpenOCD:

```
$ sudo openocd -f interface/stlink-v2.cfg -f target/stm32f4x.cfg -c init -c "reset_
↵halt" -c "flash write_image erase nuttx.bin 0x08000000"
```

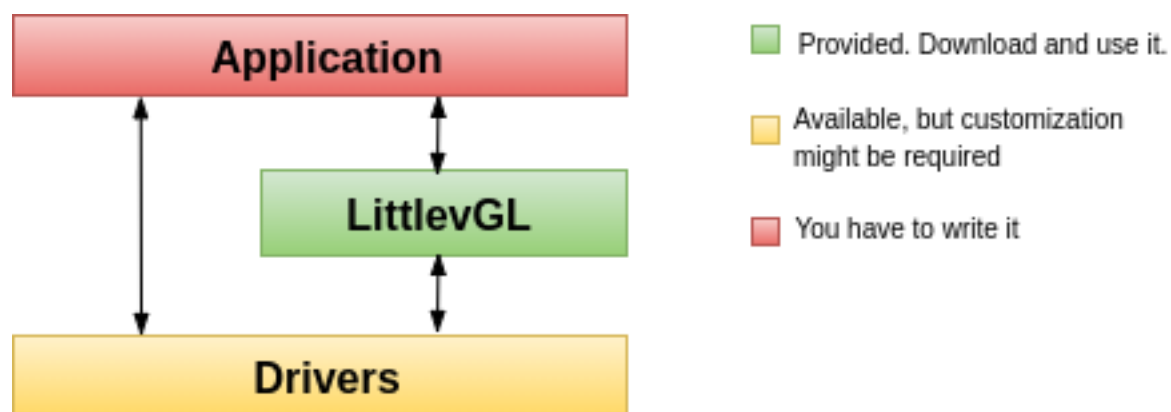
Reset the board and using the 'NSH>' terminal start the LVGL demo:

```
nsh> lvgl_demo
```

2.8.4 Where can I find more information?

- On the LVGL on LPCXpresso54628
- NuttX mailing list [Apache NuttX Mailing List](#)

3.1 System overview



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

LVGL 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.

Depending on the MCU, there are two typical hardware set-ups. One with built-in LCD/TFT driver periphery and another without it. 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

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

You can clone it or download the latest version of the library from GitHub.

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

3.2.2 Configuration file

There is a configuration header file for LVGL called **lv_conf.h**. It sets the library's basic behaviour, 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 your compiler options (e.g. **-DLV_CONF_INCLUDE_SIMPLE** for gcc compiler) and set the include path manually.

In the config file comments explain the meaning of the options. Check at least these three configuration 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

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

1. Call *lv_init()*.
2. Initialize your drivers.
3. Register the display and input devices drivers in LVGL. 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 LVGL 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];

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

There are 3 possible configurations regarding the buffer size:

1. **One buffer** LVGL 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 LVGL 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*, LVGL 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* LVGL 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 LVGL. 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`.
- **anti-aliasing** use anti-aliasing (edge smoothing). `LV_ANTIALIAS` by default from `lv_conf.h`.
- **rotated** if `1` swap `hor_res` and `ver_res`. LVGL 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.
- **screen_transp** if `1` the screen can have transparent or opaque style. `LV_COLOR_SCREEN_TRANSP` needs to be enabled in `lv_conf.h`.

To use a GPU the following callbacks can be used:

- **gpu_fill_cb** fill an area in memory with colors.
- **gpu_blend_cb** blend two memory buffers using opacity.
- **gpu_wait_cb** if any GPU function return while the GPU is still working LVGL will use this function when required to be sure GPU rendering is ready.

Note that, these functions need to draw to the memory (RAM) and not your display directly.

Some other optional callbacks to make easier and more optimal to work with monochrome, grayscale or other non-standard RGB 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 more compactly 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. `set_px_cb` is not working with **Two screen-sized buffers** display buffer configuration.
- **monitor_cb** a callback function tells how many pixels were refreshed in how much time.
- **clean_dcache_cb** a callback for cleaning any caches related to the display

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 LVGL `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 driver 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;
```

(continues on next page)

(continued from previous page)

```

    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_gpu_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;
    dest_buf += dest_width * fill_area->y1; /*Go to the first line*/

    for(y = fill_area->y1; y < fill_area->y2; y++) {
        for(x = fill_area->x1; x < fill_area->x2; x++) {
            dest_buf[x] = color;
        }
        dest_buf+=dest_width;    /*Go to the next line*/
    }
}

void my_gpu_blend_cb(lv_disp_drv_t * disp_drv, lv_color_t * dest, const lv_color_t *
↪ src, uint32_t length, lv_opa_t opa)
{
    /*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)

```

(continues on next page)

(continued from previous page)

```

{
    printf("%d px refreshed in %d ms\n", time, ms);
}

void my_clean_dcache_cb(lv_disp_drv_t * disp_drv, uint32_t)
{
    /* Example for Cortex-M (CMSIS) */
    SCB_CleanInvalidateDCache();
}

```

3.4 Input device interface

3.4.1 Types of input devices

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 LVGL 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** encoder 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.

Touchpad, mouse or any pointer

Input devices which can 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`)

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 = keyboard_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*/
}

```

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 whereby 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 = encoder_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*/
}
```

Using buttons with Encoder logic

In addition to standard encoder behavior, you can also utilise its logic to navigate(focus) and edit widgets using buttons. This is especially handy if you have only few buttons available, or you want to use other buttons in addition to encoder wheel.

You need to have 3 buttons available:

- **LV_KEY_ENTER** will simulate press or pushing of the encoder button
- **LV_KEY_LEFT** will simulate turning encoder left
- **LV_KEY_RIGHT** will simulate turning encoder right
- other keys will be passed to the focused widget

If you hold the keys it will simulate encoder click with period specified in `indev_drv.long_press_rep_time`.

```
indev_drv.type = LV_INDEV_TYPE_ENCODER;
indev_drv.read_cb = encoder_with_keys_read;

...

bool encoder_with_keys_read(lv_indev_drv_t * drv, lv_indev_data_t*data){
    data->key = last_key();           /*Get the last pressed or released key*/
                                     /* use LV_KEY_ENTER for encoder press */
    if(key_pressed()) data->state = LV_INDEV_STATE_PR;
    else {
        data->state = LV_INDEV_STATE_REL;
        /* Optionally you can also use enc_diff, if you have encoder*/
        data->enc_diff = enc_get_new_moves();
    }

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

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}, ...}`

Important: The `points_array` can't go out of scope. Either declare it as a global variable or as a static variable inside a function.

```
indev_drv.type = LV_INDEV_TYPE_BUTTON;
indev_drv.read_cb = button_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*/
}
```

3.4.2 Other features

Besides `read_cb` a `feedback_cb` callback can be also specified in `lv_indev_drv_t`. `feedback_cb` is called when any type of event is sent by the input devices. (independently from its type). It allows making feedback for the user e.g. to play a sound on `LV_EVENT_CLICK`.

The default value of the following parameters can be set in `lv_conf.h` but the default value can be overwritten in `lv_indev_drv_t`:

- **drag_limit** Number of pixels to slide before actually drag the object
- **drag_throw** Drag throw slow-down in [%]. Greater value means faster slow-down
- **long_press_time** Press time to send `LV_EVENT_LONG_PRESSED` (in milliseconds)
- **long_press_rep_time** Interval of sending `LV_EVENT_LONG_PRESSED_REPEAT` (in milliseconds)
- **read_task** pointer to the `lv_task` which reads the input device. Its parameters can be changed by `lv_task_...()` functions

Every Input device is associated with a display. By default, a new input device is added to the lastly created or the explicitly selected (using `lv_disp_set_default()`) display. The associated display is stored and can be changed in `disp` field of the driver.

3.5 Tick interface

The LVGL needs a system tick to know the elapsed time for animation and other tasks.

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

`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 operating system (e.g. on Raspberry Pi) `lv_tick_inc` can be called in a thread as below:

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

3.6 Task Handler

To handle the tasks of LVGL 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);
}
```

To learn more about task visit the [Tasks](#) section.

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*/
}
```

(continues on next page)

(continued from previous page)

```

else {
    timer_stop(); /*Stop the timer where lv_tick_inc() is called*/
    sleep();      /*Sleep the MCU*/
}
my_delay_ms(5);
}

```

You should also add below lines to your input device read function if a wake-up (press, touch or click etc.) happens:

```

lv_tick_inc(LV_DISP_DEF_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 wake-up 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 Operating system and interrupts

LVGL is **not thread-safe** by default.

However, in the following conditions it's valid to call LVGL related functions:

- In *events*. Learn more in [Events](#).
- In *lv_tasks*. Learn more in [Tasks](#).

3.8.1 Tasks and threads

If you need to use real tasks or threads, you need a mutex which should be invoked before the call of `lv_task_handler` and released after it. Also, you have to use the same mutex in other tasks and threads around every LVGL (`lv_...`) related function calls and codes. This way you can use LVGL in a real multitasking environment. Just make use of a mutex to avoid the concurrent calling of LVGL functions.

3.8.2 Interrupts

Try to avoid calling LVGL functions from the interrupts (except `lv_tick_inc()` and `lv_disp_flush_ready()`). But, if you need to do this you have to disable the interrupt which uses LVGL functions while `lv_task_handler` is running. It's a better approach to set a flag or some value and periodically check it in an `lv_task`.

3.9 Logging

LVGL has built-in *log* module to inform the user about what is happening in the library.

3.9.1 Log level

To enable logging, set `LV_USE_LOG 1` in *lv_conf.h* and set `LV_LOG_LEVEL` to one of the following values:

- `LV_LOG_LEVEL_TRACE` A lot of logs to give detailed information
- `LV_LOG_LEVEL_INFO` Log important events
- `LV_LOG_LEVEL_WARN` Log if something unwanted happened but didn't cause a problem
- `LV_LOG_LEVEL_ERROR` Only critical issue, when the system may fail
- `LV_LOG_LEVEL_NONE` Do not log anything

The events which have a higher level than the set log level will be logged too. E.g. if you `LV_LOG_LEVEL_WARN`, *errors* will be also logged.

3.9.2 Logging with printf

If your system supports `printf`, you just need to enable `LV_LOG_PRINTF` in *lv_conf.h* to send the logs with `printf`.

3.9.3 Custom log function

If you can't use `printf` or want to use a custom function to log, you can register a "logger" callback with `lv_log_register_print_cb()`.

For example:

```
void my_log_cb(lv_log_level_t level, const char * file, int line, const char * fn_
↪name, const char * dsc)
{
    /*Send the logs via serial port*/
    if(level == LV_LOG_LEVEL_ERROR) serial_send("ERROR: ");
    if(level == LV_LOG_LEVEL_WARN)  serial_send("WARNING: ");
    if(level == LV_LOG_LEVEL_INFO)  serial_send("INFO: ");
    if(level == LV_LOG_LEVEL_TRACE) serial_send("TRACE: ");

    serial_send("File: ");
    serial_send(file);

    char line_str[8];
    sprintf(line_str,"%d", line);
    serial_send("#");
    serial_send(line_str);

    serial_send(": ");
    serial_send(fn_name);
    serial_send(": ");
    serial_send(dsc);
    serial_send("\n");
}
```

(continues on next page)

(continued from previous page)

```
}  
...  
lv_log_register_print_cb(my_log_cb);
```

3.9.4 Add logs

You can also use the log module via the `LV_LOG_TRACE/INFO/WARN/ERROR(description)` functions.

OVERVIEW

4.1 Objects

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

Check all the *Object types* here.

4.1.1 Attributes

Basic attributes

All object types share some basic attributes:

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

You can set/get these 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 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 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 an 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 is 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 LVGL 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 when required and delete it when it is not visible or necessary.

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

- A pointer to the *parent* object. To create a screen give *NULL* as parent.
- Optionally, a pointer to *copy* object with the same type to copy it. This *copy* object can be *NULL* to avoid the copy operation.

All objects are referenced in C code using an **lv_obj_t** pointer as a handle. This pointer can later be used 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);
```

lv_obj_del will delete the object immediately. If for any reason you can't delete the object immediately you can use **lv_obj_del_async(obj)**. It is useful e.g. if you want to delete the parent of an object in the child's **LV_EVENT_DELETE** signal.

You can remove all the children of an object (but not the object itself) using **lv_obj_clean**:

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

4.1.3 Screens

Create screens

The screens are special objects which have no parent object. So they can be created like:

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

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

Get the active screen

There is always an active screen on each display. By default, the library creates and loads a "Base object" as a screen for each display.

To get the currently active screen use the `lv_scr_act()` function.

Load screens

To load a new screen, use `lv_scr_load(scr1)`.

Load screen with animation

A new screen can be loaded with animation too using `lv_scr_load_anim(scr, transition_type, time, delay, auto_del)`. The following transition types exist:

- `LV_SCR_LOAD_ANIM_NONE`: switch immediately after `delay` ms
- `LV_SCR_LOAD_ANIM_OVER_LEFT/RIGHT/TOP/BOTTOM` move the new screen over the other towards the given direction
- `LV_SCR_LOAD_ANIM_MOVE_LEFT/RIGHT/TOP/BOTTOM` move both the old and new screens towards the given direction
- `LV_SCR_LOAD_ANIM_FADE_ON` fade the new screen over the old screen

Setting `auto_del` to `true` will automatically delete the old screen when the animation is finished.

The new screen will become active (returned by `lv_scr_act()`) when the animations starts after `delay` time.

Handling multiple displays

Screens are created on the currently selected *default display*. The *default display* is the last registered display with `lv_disp_drv_register` or you can explicitly select a new default display using `lv_disp_set_default disp)`.

`lv_scr_act()`, `lv_scr_load()` and `lv_scr_load_anim()` operate on the default screen.

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

4.1.4 Parts

The widgets can have multiple parts. For example a *Button* has only a main part but a *Slider* is built from a background, an indicator and a knob.

The name of the parts is constructed like `LV_ + <TYPE> _PART_ <NAME>`. For example `LV_BTN_PART_MAIN` or `LV_SLIDER_PART_KNOB`. The parts are usually used when styles are add to the objects. Using parts different styles can be assigned to the different parts of the objects.

To learn more about the parts read the related section of the [Style overview](#).

4.1.5 States

The object can be in a combinations of the following states:

- **LV_STATE_DEFAULT** Normal, released
- **LV_STATE_CHECKED** Toggled or checked
- **LV_STATE_FOCUSED** Focused via keypad or encoder or clicked via touchpad/mouse
- **LV_STATE_EDITED** Edit by an encoder
- **LV_STATE_HOVERED** Hovered by mouse (not supported now)
- **LV_STATE_PRESSED** Pressed
- **LV_STATE_DISABLED** Disabled or inactive

The states are usually automatically changed by the library as the user presses, releases, focuses etc an object. However, the states can be changed manually too. To completely overwrite the current state use `lv_obj_set_state(obj, part, LV_STATE...)`. To set or clear given state (but leave to other states untouched) use `lv_obj_add/clear_state(obj, part, LV_STATE...)` In both cases Ored state values can be used as well. E.g. `lv_obj_set_state(obj, part, LV_STATE_PRESSED | LV_PRESSED_CHECKED)`.

To learn more about the states read the related section of the [Style overview](#).

4.2 Layers

4.2.1 Order of creation

By default, LVGL draws old objects on the background and new objects on the foreground.

For example, assume we added a button to a parent object named `button1` and then another button named `button2`. Then `button1` (with its child object(s)) will be in the background and can be covered by `button2` 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*/
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 LVGL will automatically bring the object to the foreground. It works similarly to a typical GUI on a PC. When a window in the background is clicked, it will come to the foreground automatically.
- Use `lv_obj_move_foreground(obj)` to explicitly tell the library to bring an object to the foreground. Similarly, use `lv_obj_move_background(obj)` to 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 layers

LVGL uses two special layers named as `layer_top` and `layer_sys`. Both are visible and common on all screens of a display. **They are not, however, shared among multiple physical displays.** The `layer_top` is always on top of the default screen (`lv_scr_act()`), and `layer_sys` is on top of `layer_top`.

The `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);
```

The `layer_sys` is also using for similar purpose on LVGL. For example, it places the mouse cursor there to be sure it's always visible.

4.3 Events

Events are triggered in LVGL when something happens which might be interesting to the user, e.g. if an object:

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

The user can assign a callback function to an object to see these events. In 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:
```

(continues on next page)

(continued from previous page)

```

        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

All objects (such as Buttons/Labels/Sliders etc.) receive these generic events regardless of their type.

Related to the input devices

These are sent when an object is pressed/released etc. by the user. They are used not only for *Pointers* but can be used for *Keypad*, *Encoder* and *Button* input devices as well. 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** The input device is still being pressed but is no longer on the object
- **LV_EVENT_SHORT_CLICKED** Released before **LV_INDEV_LONG_PRESS_TIME** 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.

Related to pointer

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")

Related to keypad and encoder

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

- **LV_EVENT_KEY** A *Key* is sent to the object. Typically when it was pressed or repeated after a long press. The key can be retrived by `uint32_t * key = lv_event_get_data()`
- **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 particular 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.

Visit particular *Object type's documentation* to understand which events are used by an object type.

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 information, see the particular *Object type's documentation*. To get the custom data in the event callback use `lv_event_get_data()`.

The type of the custom data depends on the sending object but if it's a

- single number then it's `uint32_t *` or `int32_t *`
- text then `char *` or `const char *`

4.3.3 Send events manually

Arbitrary events

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

For example, it can be used to manually close a message box by simulating a button press (although there are simpler ways of doing this):

```
/*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);
```

Refresh event

`LV_EVENT_REFRESH` is special event because it's designed to be used by the user to notify an object to refresh itself. Some examples:

- notify a label to refresh its text according to one or more variables (e.g. current time)
- refresh a label when the language changes
- enable a button if some conditions are met (e.g. the correct PIN is entered)
- add/remove styles to/from an object if a limit is exceeded, etc

To simplest way to handle similar cases is utilizing the following functions.

`lv_event_send_refresh(obj)` is just a wrapper to `lv_event_send(obj, LV_EVENT_REFRESH, NULL)`. So it simply sends an `LV_EVENT_REFRESH` to an object.

`lv_event_send_refresh_recursive(obj)` sends `LV_EVENT_REFRESH` event to an object and all of its children. If `NULL` is passed as parameter all objects of all displays will be refreshed.

4.4 Styles

Styles are used to set the appearance of the objects. Styles in lvgl are heavily inspired by CSS. The concept in nutshell is the following:

- A style is an `lv_style_t` variable which can hold properties, for example border width, text color and so on. It's similar to `class` in CSS.
- Not all properties have to be specified. Unspecified properties will use a default value.
- Styles can be assigned to objects to change their appearance.

- A style can be used by any number of objects.
- Styles can be cascaded which means multiple styles can be assigned to an object and each style can have different properties. For example `style_btn` can result in a default gray button and `style_btn_red` can add only a `background-color=red` to overwrite the background color.
- Later added styles have higher precedence. It means if a property is specified in two styles the later added will be used.
- Some properties (e.g. text color) can be inherited from the parent(s) if it's not specified in the object.
- Objects can have local styles that have higher precedence than "normal" styles.
- Unlike CSS (where pseudo-classes describes different states, e.g. `:hover`), in lvgl a property is assigned to a given state. (I.e. not the "class" is related to state but every single property has a state)
- Transitions can be applied when the object changes state.

4.4.1 States

The objects can be in the following states:

- **LV_STATE_DEFAULT** (0x00): Normal, released
- **LV_STATE_CHECKED** (0x01): Toggled or checked
- **LV_STATE_FOCUSED** (0x02): Focused via keypad or encoder or clicked via touchpad/mouse
- **LV_STATE_EDITED** (0x04): Edit by an encoder
- **LV_STATE_HOVERED** (0x08): Hovered by mouse (not supported now)
- **LV_STATE_PRESSED** (0x10): Pressed
- **LV_STATE_DISABLED** (0x20): Disabled or inactive

Combination of states is also possible, for example `LV_STATE_FOCUSED | LV_STATE_PRESSED`.

The style properties can be defined in every state and state combination. For example, setting a different background color for default and pressed state. If a property is not defined in a state the best matching state's property will be used. Typically it means the property with `LV_STATE_DEFAULT` state. If the property is not set even for the default state the default value will be used. (See later)

But what does the "best matching state's property" really means? States have a precedence which is shown by their value (see in the above list). A higher value means higher precedence. To determine which state's property to use let's use an example. Let's see the background color is defined like this:

- **LV_STATE_DEFAULT**: white
- **LV_STATE_PRESSED**: gray
- **LV_STATE_FOCUSED**: red

1. By the default the object is in default state, so it's a simple case: the property is perfectly defined in the object's current state as white
2. When the object is pressed there are 2 related properties: default with white (default is related to every state) and pressed with gray. The pressed state has 0x10 precedence which is higher than the default state's 0x00 precedence, so gray color will be used.
3. When the object is focused the same thing happens as in pressed state and red color will be used. (Focused state has higher precedence than default state).
4. When the object is focused and pressed both gray and red would work, but the pressed state has higher precedence than focused so gray color will be used.

5. It's possible to set e.g. rose color for `LV_STATE_PRESSED | LV_STATE_FOCUSED`. In this case, this combined state has $0x02 + 0x10 = 0x12$ precedence, which is higher than the pressed state's precedence so the rose color would be used.
6. When the object is checked there is no property to set the background color for this state. So in lack of a better option, the object remains white from the default state's property.

Some practical notes:

- If you want to set a property for all state (e.g. red background color) just set it for the default state. If the object can't find a property for its current state it will fall back to the default state's property.
- Use ORed states to describe the properties for complex cases. (E.g. pressed + checked + focused)
- It might be a good idea to use different style elements for different states. For example, finding background colors for released, pressed, checked + pressed, focused, focused + pressed, focused + pressed + checked, etc states is quite difficult. Instead, for example, use the background color for pressed and checked states and indicate the focused state with a different border color.

4.4.2 Cascading styles

It's not required to set all the properties in one style. It's possible to add more styles to an object and let the later added style to modify or extend the properties in the other styles. For example, create a general gray button style and create a new one for red buttons where only the new background color is set.

It's the same concept when in CSS all the used classes are listed like `<div class=".btn .btn-red">`.

The later added styles have higher precedence over the earlier ones. So in the gray/red button example above, the normal button style should be added first and the red style second. However, the precedence coming from states is still taken into account. So let's examine the following case:

- the basic button style defines dark-gray color for default state and light-gray color pressed state
- the red button style defines the background color as red only in the default state

In this case, when the button is released (it's in default state) it will be red because a perfect match is found in the lastly added style (red style). When the button is pressed the light-gray color is a better match because it describes the current state perfectly, so the button will be light-gray.

4.4.3 Inheritance

Some properties (typically that are related to texts) can be inherited from the parent object's styles. Inheritance is applied only if the given property is not set in the object's styles (even in default state). In this case, if the property is inheritable, the property's value will be searched in the parent too until a part can tell a value for the property. The parents will use their own state to tell the value. So if a button is pressed, and text color comes from here, the pressed text color will be used.

4.4.4 Parts

Objects can have *parts* which can have their own style. For example a *page* has four parts:

- Background
- Scrollable
- Scrollbar
- Edge flash

There is three types of object parts **main**, **virtual** and **real**.

The main part is usually the background and largest part of the object. Some object has only a main part. For example, a button has only a background.

The virtual parts are additional parts just drawn on the fly to the main part. There is no "real" object behind them. For example, the page's scrollbar is not a real object, it's just drawn when the page's background is drawn. The virtual parts always have the same state as the main part. If the property can be inherited, the main part will be also considered before going to the parent.

The real parts are real objects created and managed by the main object. For example, the page's scrollable part is real object. Real parts can be in different state than the main part.

To see which parts an object has visit their documentation page.

4.4.5 Initialize styles and set/get properties

Styles are stored in `lv_style_t` variables. Style variables should be **static**, global or dynamically allocated. In other words they can not be local variables in functions which are destroyed when the function exists. Before using a style it should be initialized with `lv_style_init(&my_style)`. After initializing the style properties can be set added to it. Property set functions looks like this: `lv_style_set_<property_name>(&style, <state>, <value>);` For example the *above mentioned* example looks like this:

```
static lv_style_t style1;
lv_style_set_bg_color(&style1, LV_STATE_DEFAULT, LV_COLOR_WHITE);
lv_style_set_bg_color(&style1, LV_STATE_PRESSED, LV_COLOR_GRAY);
lv_style_set_bg_color(&style1, LV_STATE_FOCUSED, LV_COLOR_RED);
lv_style_set_bg_color(&style1, LV_STATE_FOCUSED | LV_STATE_PRESSED, lv_color_
↪ hex(0xf88));
```

It's possible to copy a style with `lv_style_copy(&style_destination, &style_source)`. After copy properties still can be added freely.

To remove a property use:

```
lv_style_remove_prop(&style, LV_STYLE_BG_COLOR | (LV_STATE_PRESSED << LV_STYLE_STATE_
↪ POS));
```

To get the value from style in a given state functions with the following prototype are available: `_lv_style_get_color/int/opa/ptr(&style, <prop>, <result buf>);`. The best matching property will be selected and it's precedence will be returned. -1 will be returned if the property is not found.

The form of the function (`...color/int/opa/ptr`) should be used according to the type of `<prop>`.

For example:


```
lv_color_t color;
int16_t res;
res = _lv_style_get_color(&style1, LV_STYLE_BG_COLOR | (LV_STATE_PRESSED << LV_STYLE_
↪STATE_POS), &color);
if(res >= 0) {
    //the bg_color is loaded into `color`
}
```

To reset a style (free all it's data) use

```
lv_style_reset(&style);
```

4.4.6 Managing style list

A style on its own not that useful. It should be assigned to an object to take its effect. Every part of the objects stores a *style list* which is the list of assigned styles.

To add a style to an object use `lv_obj_add_style(obj, <part>, &style)` For example:

```
lv_obj_add_style(btn, LV_BTN_PART_MAIN, &btn);    /*Default button style*/
lv_obj_add_style(btn, LV_BTN_PART_MAIN, &btn_red); /*Overwrite only a some colors to
↪red*/
```

An objects style list can be reset with `lv_obj_reset_style_list(obj, <part>)`

If a style which is already assigned to an object changes (i.e. one of it's property is set to a new value) the objects using that style should be notified with `lv_obj_refresh_style(obj)`

To get a final value of property, including cascading, inheritance, local styles and transitions (see below), get functions like this can be used: `lv_obj_get_style_<property_name>(obj, <part>)`. These functions uses the object's current state and if no better candidate returns a default value. For example:

```
lv_color_t color = lv_obj_get_style_bg_color(btn, LV_BTN_PART_MAIN);
```

4.4.7 Local styles

In the object's style lists, so-called local properties can be stored as well. It's the same concept than CSS's `<div style="color:red">`. The local style is the same as a normal style, but it belongs only to a given object and can not be shared with other objects. To set a local property use functions like `lv_obj_set_style_local_<property_name>(obj, <part>, <state>, <value>)`; For example:

```
lv_obj_set_style_local_bg_color(btn, LV_BTN_PART_MAIN, LV_STATE_DEFAULT, LV_COLOR_
↪RED);
```

4.4.8 Transitions

By default, when an object changes state (e.g. it's pressed) the new properties from the new state are set immediately. However, with transitions it's possible to play an animation on state change. For example, on pressing a button its background color can be animated to the pressed color over 300 ms.

The parameters of the transitions are stored in the styles. It's possible to set

- the time of the transition
- the delay before starting the transition
- the animation path (also known as timing function)
- the properties to animate

The transition properties can be defined for each state. For example, setting 500 ms transition time in default state will mean that when the object goes to default state 500 ms transition time will be applied. Setting 100 ms transition time in the pressed state will mean a 100 ms transition time when going to pressed state. So this example configuration will result in fast going to pressed state and slow going back to default.

4.4.9 Properties

The following properties can be used in the styles.

Mixed properties

- **radius** (`lv_style_int_t`): Set the radius of the background. 0: no radius, `LV_RADIUS_CIRCLE`: maximal radius. Default value: 0.
- **clip_corner** (`bool`): **true**: enable to clip the overflowed content on the rounded (radius > 0) corners. Default value: **false**.
- **size** (`lv_style_int_t`): Size of internal elements of the widgets. See the documentation of the widgets if this property is used or not. Default value: `LV_DPI / 20`.
- **transform_width** (`lv_style_int_t`): Make the object wider on both sides with this value. Default value: 0.
- **transform_height** (`lv_style_int_t`): Make the object higher on both sides with this value. Default value: 0.
- **transform_angle** (`lv_style_int_t`): Rotate the image-like objects. It's unit is 0.1 deg, for 45 deg use 450. Default value: 0.
- **transform_zoom** (`lv_style_int_t`): Zoom image-like objects. 256 (or `LV_IMG_ZOOM_NONE`) for normal size, 128 half size, 512 double size, and so on. Default value: `LV_IMG_ZOOM_NONE`.
- **opa_scale** (`lv_style_int_t`): Inherited. Scale down all opacity values of the object by this factor. As it's inherited the children objects will be affected too. Default value: `LV_OPA_COVER`.

Padding and margin properties

Padding sets the space on the inner sides of the edges. It means "I don't want my children too close to my sides, so keep this space". *Padding inner* set the "gap" between the children. *Margin* sets the space on the outer side of the edges. It means "I want this space around me".

These properties are typically used by *Container* object if *layout* or *auto fit* is enabled. However other widgets also use them to set spacing. See the documentation of the widgets for the details.

- **pad_top** (`lv_style_int_t`): Set the padding on the top. Default value: 0.
- **pad_bottom** (`lv_style_int_t`): Set the padding on the bottom. Default value: 0.
- **pad_left** (`lv_style_int_t`): Set the padding on the left. Default value: 0.
- **pad_right** (`lv_style_int_t`): Set the padding on the right. Default value: 0.
- **pad_inner** (`lv_style_int_t`): Set the padding inside the object between children. Default value: 0.
- **margin_top** (`lv_style_int_t`): Set the margin on the top. Default value: 0.
- **margin_bottom** (`lv_style_int_t`): Set the margin on the bottom. Default value: 0.
- **margin_left** (`lv_style_int_t`): Set the margin on the left. Default value: 0.
- **margin_right** (`lv_style_int_t`): Set the margin on the right. Default value: 0.

Background properties

The background is a simple rectangle which can have gradient and **radius** rounding.

- **bg_color** (`lv_color_t`) Specifies the color of the background. Default value: `LV_COLOR_WHITE`.
- **bg_opa** (`lv_opa_t`) Specifies opacity of the background. Default value: `LV_OPA TRANSP`.
- **bg_grad_color** (`lv_color_t`) Specifies the color of the background's gradient. The color on the right or bottom is **bg_grad_dir** != `LV_GRAD_DIR_NONE`. Default value: `LV_COLOR_WHITE`.
- **bg_main_stop** (`uint8_t`): Specifies where should the gradient start. 0: at left/top most position, 255: at right/bottom most position. Default value: 0.
- **bg_grad_stop** (`uint8_t`): Specifies where should the gradient stop. 0: at left/top most position, 255: at right/bottom most position. Default value: 255.
- **bg_grad_dir** (`lv_grad_dir_t`) Specifies the direction of the gradient. Can be `LV_GRAD_DIR_NONE/HOR/VER`. Default value: `LV_GRAD_DIR_NONE`.
- **bg_blend_mode** (`lv_blend_mode_t`): Set the blend mode the background. Can be `LV_BLEND_MODE_NORMAL/ADDITIVE/SUBTRACTIVE`). Default value: `LV_BLEND_MODE_NORMAL`.



```
#include "../../lv_examples.h"

/**
 * Using the background style properties
 */
void lv_ex_style_1(void)
{
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_radius(&style, LV_STATE_DEFAULT, 5);

    /*Make a gradient*/
    lv_style_set_bg_opa(&style, LV_STATE_DEFAULT, LV_OPA_COVER);
    lv_style_set_bg_color(&style, LV_STATE_DEFAULT, LV_COLOR_SILVER);
    lv_style_set_bg_grad_color(&style, LV_STATE_DEFAULT, LV_COLOR_BLUE);
    lv_style_set_bg_grad_dir(&style, LV_STATE_DEFAULT, LV_GRAD_DIR_VER);

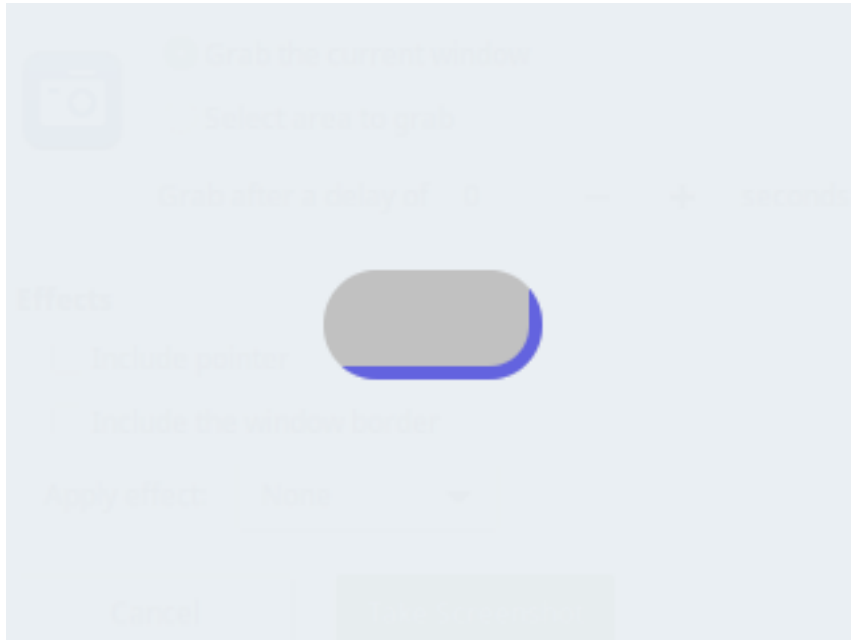
    /*Shift the gradient to the bottom*/
    lv_style_set_bg_main_stop(&style, LV_STATE_DEFAULT, 128);
    lv_style_set_bg_grad_stop(&style, LV_STATE_DEFAULT, 192);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_scr_act(), NULL);
    lv_obj_add_style(obj, LV_OBJ_PART_MAIN, &style);
    lv_obj_align(obj, NULL, LV_ALIGN_CENTER, 0, 0);
}
```

Border properties

The border is drawn on top of the *background*. It has **radius** rounding.

- **border_color** (`lv_color_t`) Specifies the color of the border. Default value: `LV_COLOR_BLACK`.
- **border_opa** (`lv_opa_t`) Specifies opacity of the border. Default value: `LV_OPA_COVER`.
- **border_width** (`lv_style_int_t`): Set the width of the border. Default value: 0.
- **border_side** (`lv_border_side_t`) Specifies which sides of the border to draw. Can be `LV_BORDER_SIDE_NONE/LEFT/RIGHT/TOP/BOTTOM/FULL`. ORed values are also possible. Default value: `LV_BORDER_SIDE_FULL`.
- **border_post** (`bool`): If `true` the border will be drawn after all children have been drawn. Default value: `false`.
- **border_blend_mode** (`lv_blend_mode_t`): Set the blend mode of the border. Can be `LV_BLEND_MODE_NORMAL/ADDITIVE/SUBTRACTIVE`). Default value: `LV_BLEND_MODE_NORMAL`.



```
#include "../../lv_examples.h"

/**
 * Using the border style properties
 */
void lv_ex_style_2(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, LV_STATE_DEFAULT, 20);
    lv_style_set_bg_opa(&style, LV_STATE_DEFAULT, LV_OPA_COVER);
    lv_style_set_bg_color(&style, LV_STATE_DEFAULT, LV_COLOR_SILVER);

    /*Add border to the bottom+right*/
    lv_style_set_border_color(&style, LV_STATE_DEFAULT, LV_COLOR_BLUE);
}
```

(continues on next page)

(continued from previous page)

```

lv_style_set_border_width(&style, LV_STATE_DEFAULT, 5);
lv_style_set_border_opa(&style, LV_STATE_DEFAULT, LV_OPA_50);
lv_style_set_border_side(&style, LV_STATE_DEFAULT, LV_BORDER_SIDE_BOTTOM | LV_
↪BORDER_SIDE_RIGHT);

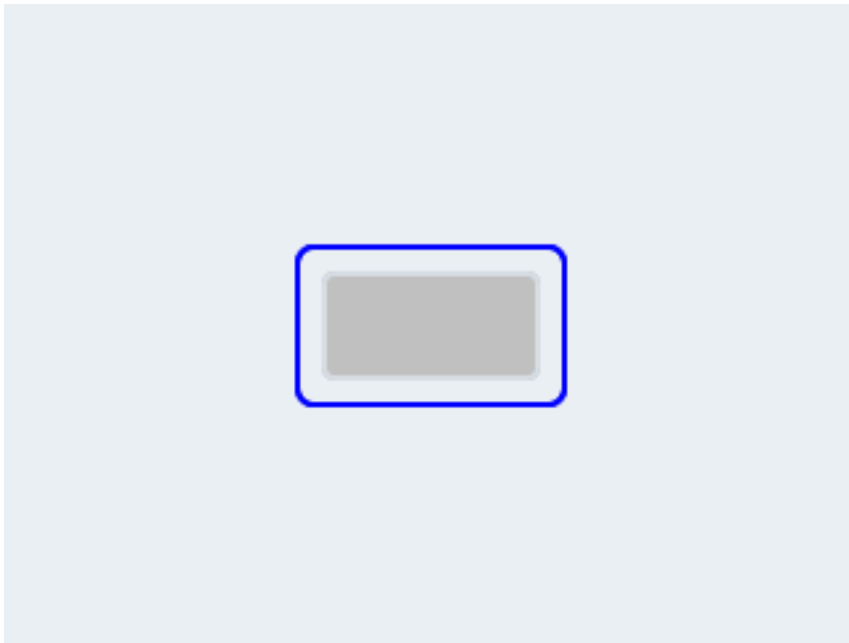
/*Create an object with the new style*/
lv_obj_t * obj = lv_obj_create(lv_scr_act(), NULL);
lv_obj_add_style(obj, LV_OBJ_PART_MAIN, &style);
lv_obj_align(obj, NULL, LV_ALIGN_CENTER, 0, 0);
}

```

Outline properties

The outline is similar to *border* but is drawn outside of the object.

- **outline_color** (`lv_color_t`) Specifies the color of the outline. Default value: `LV_COLOR_BLACK`.
- **outline_opa** (`lv_opa_t`) Specifies opacity of the outline. Default value: `LV_OPA_COVER`.
- **outline_width** (`lv_style_int_t`): Set the width of the outline. Default value: 0.
- **outline_pad** (`lv_style_int_t`) Set the space between the object and the outline. Default value: 0.
- **outline_blend_mode** (`lv_blend_mode_t`): Set the blend mode of the outline. Can be `LV_BLEND_MODE_NORMAL/ADDITIVE/SUBTRACTIVE`). Default value: `LV_BLEND_MODE_NORMAL`.



```

#include "../../lv_examples.h"

/**
 * Using the outline style properties
 */
void lv_ex_style_3(void)

```

(continues on next page)

(continued from previous page)

```

{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, LV_STATE_DEFAULT, 5);
    lv_style_set_bg_opa(&style, LV_STATE_DEFAULT, LV_OPA_COVER);
    lv_style_set_bg_color(&style, LV_STATE_DEFAULT, LV_COLOR_SILVER);

    /*Add outline*/
    lv_style_set_outline_width(&style, LV_STATE_DEFAULT, 2);
    lv_style_set_outline_color(&style, LV_STATE_DEFAULT, LV_COLOR_BLUE);
    lv_style_set_outline_pad(&style, LV_STATE_DEFAULT, 8);

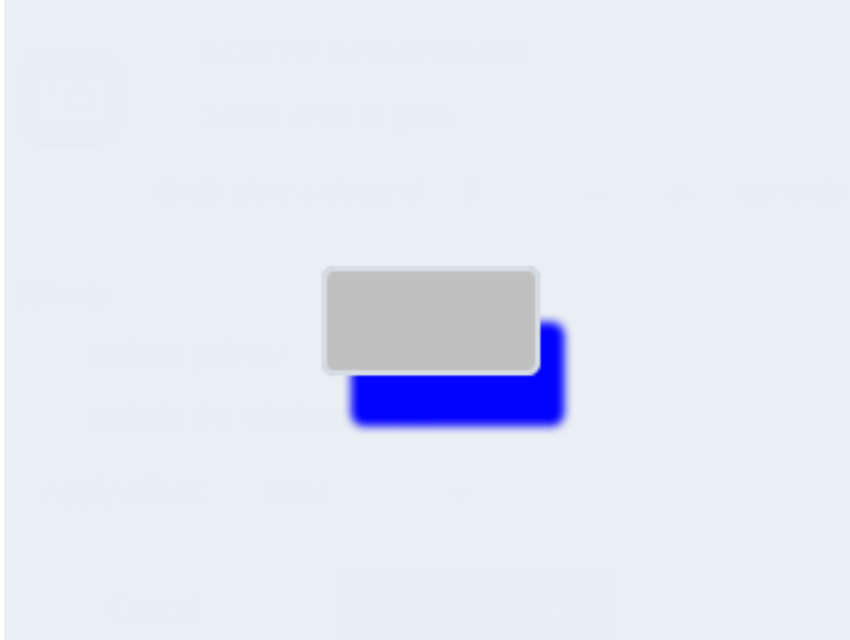
    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_scr_act(), NULL);
    lv_obj_add_style(obj, LV_OBJ_PART_MAIN, &style);
    lv_obj_align(obj, NULL, LV_ALIGN_CENTER, 0, 0);
}

```

Shadow properties

The shadow is a blurred area under the object.

- **shadow_color** (`lv_color_t`) Specifies the color of the shadow. Default value: `LV_COLOR_BLACK`.
- **shadow_opa** (`lv_opa_t`) Specifies opacity of the shadow. Default value: `LV_OPA_TRANSP`.
- **shadow_width** (`lv_style_int_t`): Set the width (blur size) of the outline. Default value: 0.
- **shadow_ofs_x** (`lv_style_int_t`): Set the an X offset for the shadow. Default value: 0.
- **shadow_ofs_y** (`lv_style_int_t`): Set the an Y offset for the shadow. Default value: 0.
- **shadow_spread** (`lv_style_int_t`): make the shadow larger than the background in every direction by this value. Default value: 0.
- **shadow_blend_mode** (`lv_blend_mode_t`): Set the blend mode of the shadow. Can be `LV_BLEND_MODE_NORMAL/ADDITIVE/SUBTRACTIVE`). Default value: `LV_BLEND_MODE_NORMAL`.



```
#include "../../lv_examples.h"

/**
 * Using the Shadow style properties
 */
void lv_ex_style_4(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, LV_STATE_DEFAULT, 5);
    lv_style_set_bg_opa(&style, LV_STATE_DEFAULT, LV_OPA_COVER);
    lv_style_set_bg_color(&style, LV_STATE_DEFAULT, LV_COLOR_SILVER);

    /*Add a shadow*/
    lv_style_set_shadow_width(&style, LV_STATE_DEFAULT, 8);
    lv_style_set_shadow_color(&style, LV_STATE_DEFAULT, LV_COLOR_BLUE);
    lv_style_set_shadow_ofs_x(&style, LV_STATE_DEFAULT, 10);
    lv_style_set_shadow_ofs_y(&style, LV_STATE_DEFAULT, 20);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_scr_act(), NULL);
    lv_obj_add_style(obj, LV_OBJ_PART_MAIN, &style);
    lv_obj_align(obj, NULL, LV_ALIGN_CENTER, 0, 0);
}
```


Pattern properties

The pattern is an image (or symbol) drawn in the middle of the background or repeated to fill the whole background.

- **pattern_image** (`const void *`): Pointer to an `lv_img_dsc_t` variable, a path to an image file or a symbol. Default value: `NULL`.
- **pattern_opa** (`lv_opa_t`): Specifies opacity of the pattern. Default value: `LV_OPA_COVER`.
- **pattern_recolor** (`lv_color_t`): Mix this color to the pattern image. In case of symbols (texts) it will be the text color. Default value: `LV_COLOR_BLACK`.
- **pattern_recolor_opa** (`lv_opa_t`): Intensity of recoloring. Default value: `LV_OPA_TRANSP` (no recoloring).
- **pattern_repeat** (`bool`): `true`: the pattern will be repeated as a mosaic. `false`: place the pattern in the middle of the background. Default value: `false`.
- **pattern_blend_mode** (`lv_blend_mode_t`): Set the blend mode of the pattern. Can be `LV_BLEND_MODE_NORMAL`/`ADDITIVE`/`SUBTRACTIVE`). Default value: `LV_BLEND_MODE_NORMAL`.



```
#include "../../lv_examples.h"

/**
 * Using the pattern style properties
 */
void lv_ex_style_5(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, LV_STATE_DEFAULT, 5);
    lv_style_set_bg_opa(&style, LV_STATE_DEFAULT, LV_OPA_COVER);
    lv_style_set_bg_color(&style, LV_STATE_DEFAULT, LV_COLOR_SILVER);
}
```

(continues on next page)

(continued from previous page)

```

/*Add a repeating pattern*/
lv_style_set_pattern_image(&style, LV_STATE_DEFAULT, LV_SYMBOL_OK);
lv_style_set_pattern_recolor(&style, LV_STATE_DEFAULT, LV_COLOR_BLUE);
lv_style_set_pattern_opa(&style, LV_STATE_DEFAULT, LV_OPA_50);
lv_style_set_pattern_repeat(&style, LV_STATE_DEFAULT, true);

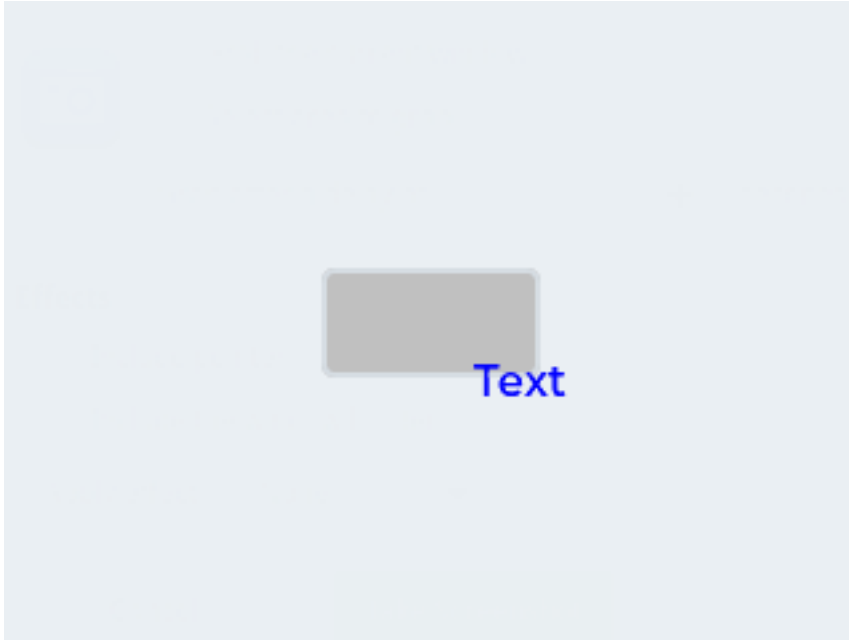
/*Create an object with the new style*/
lv_obj_t * obj = lv_obj_create(lv_scr_act(), NULL);
lv_obj_add_style(obj, LV_OBJ_PART_MAIN, &style);
lv_obj_align(obj, NULL, LV_ALIGN_CENTER, 0, 0);
}

```

Value properties

Value is an arbitrary text drawn to the background. It can be a lightweighted replacement of creating label objects.

- **value_str** (const char *): Pointer to text to display. Only the pointer is saved! (Don't use local variable with lv_style_set_value_str, instead use static, global or dynamically allocated data). Default value: NULL.
- **value_color** (lv_color_t): Color of the text. Default value: LV_COLOR_BLACK.
- **value_opa** (lv_opa_t): Opacity of the text. Default value: LV_OPA_COVER.
- **value_font** (const lv_font_t *): Pointer to font of the text. Default value: NULL.
- **value_letter_space** (lv_style_int_t): Letter space of the text. Default value: 0.
- **value_line_space** (lv_style_int_t): Line space of the text. Default value: 0.
- **value_align** (lv_align_t): Alignment of the text. Can be LV_ALIGN_.... Default value: LV_ALIGN_CENTER.
- **value_ofs_x** (lv_style_int_t): X offset from the original position of the alignment. Default value: 0.
- **value_ofs_y** (lv_style_int_t): Y offset from the original position of the alignment. Default value: 0.
- **value_blend_mode** (lv_blend_mode_t): Set the blend mode of the text. Can be LV_BLEND_MODE_NORMAL/ADDITIVE/SUBTRACTIVE). Default value: LV_BLEND_MODE_NORMAL.



```
#include "../../lv_examples.h"

/**
 * Using the value style properties
 */
void lv_ex_style_6(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, LV_STATE_DEFAULT, 5);
    lv_style_set_bg_opa(&style, LV_STATE_DEFAULT, LV_OPA_COVER);
    lv_style_set_bg_color(&style, LV_STATE_DEFAULT, LV_COLOR_SILVER);

    /*Add a value text properties*/
    lv_style_set_value_color(&style, LV_STATE_DEFAULT, LV_COLOR_BLUE);
    lv_style_set_value_align(&style, LV_STATE_DEFAULT, LV_ALIGN_IN_BOTTOM_RIGHT);
    lv_style_set_value_ofs_x(&style, LV_STATE_DEFAULT, 10);
    lv_style_set_value_ofs_y(&style, LV_STATE_DEFAULT, 10);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_scr_act(), NULL);
    lv_obj_add_style(obj, LV_OBJ_PART_MAIN, &style);
    lv_obj_align(obj, NULL, LV_ALIGN_CENTER, 0, 0);

    /*Add a value text to the local style. This way every object can have different
    ↪ text*/
    lv_obj_set_style_local_value_str(obj, LV_OBJ_PART_MAIN, LV_STATE_DEFAULT, "Text");
}
```

Text properties

Properties for textual object.

- `text_color` (`lv_color_t`): Color of the text. Default value: `LV_COLOR_BLACK`.
- `text_opa` (`lv_opa_t`): Opacity of the text. Default value: `LV_OPA_COVER`.
- `text_font` (`const lv_font_t *`): Pointer to font of the text. Default value: `NULL`.
- `text_letter_space` (`lv_style_int_t`): Letter space of the text. Default value: 0.
- `text_line_space` (`lv_style_int_t`): Line space of the text. Default value: 0.
- `text_decor` (`lv_text_decor_t`): Add text decoration. Can be `LV_TEXT_DECOR_NONE/UNDERLINE/STRIKETHROUGH`. Default value: `LV_TEXT_DECOR_NONE`.
- `text_sel_color` (`lv_color_t`): Set background color of text selection. Default value: `LV_COLOR_BLACK`.
- `text_blend_mode` (`lv_blend_mode_t`): Set the blend mode of the text. Can be `LV_BLEND_MODE_NORMAL/ADDITIVE/SUBTRACTIVE`. Default value: `LV_BLEND_MODE_NORMAL`.



```
#include "../../lv_examples.h"

/**
 * Using the text style properties
 */
void lv_ex_style_7(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    lv_style_set_radius(&style, LV_STATE_DEFAULT, 5);
    lv_style_set_bg_opa(&style, LV_STATE_DEFAULT, LV_OPA_COVER);
    lv_style_set_bg_color(&style, LV_STATE_DEFAULT, LV_COLOR_SILVER);
    lv_style_set_border_width(&style, LV_STATE_DEFAULT, 2);
    lv_style_set_border_color(&style, LV_STATE_DEFAULT, LV_COLOR_BLUE);
}
```

(continues on next page)

(continued from previous page)

```

lv_style_set_pad_top(&style, LV_STATE_DEFAULT, 10);
lv_style_set_pad_bottom(&style, LV_STATE_DEFAULT, 10);
lv_style_set_pad_left(&style, LV_STATE_DEFAULT, 10);
lv_style_set_pad_right(&style, LV_STATE_DEFAULT, 10);

lv_style_set_text_color(&style, LV_STATE_DEFAULT, LV_COLOR_BLUE);
lv_style_set_text_letter_space(&style, LV_STATE_DEFAULT, 5);
lv_style_set_text_line_space(&style, LV_STATE_DEFAULT, 20);
lv_style_set_text_decor(&style, LV_STATE_DEFAULT, LV_TEXT_DECOR_UNDERLINE);

/*Create an object with the new style*/
lv_obj_t * obj = lv_label_create(lv_scr_act(), NULL);
lv_obj_add_style(obj, LV_LABEL_PART_MAIN, &style);
lv_label_set_text(obj, "Text of\n"
                    "a label");
lv_obj_align(obj, NULL, LV_ALIGN_CENTER, 0, 0);
}

```

Line properties

Properties of lines.

- **line_color** (`lv_color_t`): Color of the line. Default value: `LV_COLOR_BLACK`
- **line_opa** (`lv_opa_t`): Opacity of the line. Default value: `LV_OPA_COVER`
- **line_width** (`lv_style_int_t`): Width of the line. Default value: 0.
- **line_dash_width** (`lv_style_int_t`): Width of dash. Dashing is drawn only for horizontal or vertical lines. 0: disable dash. Default value: 0.
- **line_dash_gap** (`lv_style_int_t`): Gap between two dash line. Dashing is drawn only for horizontal or vertical lines. 0: disable dash. Default value: 0.
- **line_rounded** (`bool`): `true`: draw rounded line endings. Default value: `false`.
- **line_blend_mode** (`lv_blend_mode_t`): Set the blend mode of the line. Can be `LV_BLEND_MODE_NORMAL`/`ADDITIVE`/`SUBTRACTIVE`). Default value: `LV_BLEND_MODE_NORMAL`.



```
#include "../../lv_examples.h"

/**
 * Using the line style properties
 */
void lv_ex_style_8(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    lv_style_set_line_color(&style, LV_STATE_DEFAULT, LV_COLOR_GRAY);
    lv_style_set_line_width(&style, LV_STATE_DEFAULT, 6);
    lv_style_set_line_rounded(&style, LV_STATE_DEFAULT, true);
#ifdef LV_USE_LINE
    /*Create an object with the new style*/
    lv_obj_t * obj = lv_line_create(lv_scr_act(), NULL);
    lv_obj_add_style(obj, LV_LINE_PART_MAIN, &style);

    static lv_point_t p[] = {{10, 30}, {30, 50}, {100, 0}};
    lv_line_set_points(obj, p, 3);

    lv_obj_align(obj, NULL, LV_ALIGN_CENTER, 0, 0);
#endif
}
```

Image properties

Properties of image.

- **image_recolor** (**lv_color_t**): Mix this color to the pattern image. In case of symbols (texts) it will be the text color. Default value: **LV_COLOR_BLACK**
- **image_recolor_opa** (**lv_opa_t**): Intensity of recoloring. Default value: **LV_OPA_TRANSP** (no recoloring). Default value: **LV_OPA_TRANSP**
- **image_opa** (**lv_opa_t**): Opacity of the image. Default value: **LV_OPA_COVER**
- **image_blend_mode** (**lv_blend_mode_t**): Set the blend mode of the image. Can be **LV_BLEND_MODE_NORMAL**/**ADDITIVE**/**SUBTRACTIVE**). Default value: **LV_BLEND_MODE_NORMAL**.



```
#include "../../lv_examples.h"

/**
 * Using the image style properties
 */
void lv_ex_style_9(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, LV_STATE_DEFAULT, 5);
    lv_style_set_bg_opa(&style, LV_STATE_DEFAULT, LV_OPA_COVER);
    lv_style_set_bg_color(&style, LV_STATE_DEFAULT, LV_COLOR_SILVER);
    lv_style_set_border_width(&style, LV_STATE_DEFAULT, 2);
    lv_style_set_border_color(&style, LV_STATE_DEFAULT, LV_COLOR_BLUE);

    lv_style_set_pad_top(&style, LV_STATE_DEFAULT, 10);
    lv_style_set_pad_bottom(&style, LV_STATE_DEFAULT, 10);
    lv_style_set_pad_left(&style, LV_STATE_DEFAULT, 10);
    lv_style_set_pad_right(&style, LV_STATE_DEFAULT, 10);
}
```

(continues on next page)

(continued from previous page)

```

lv_style_set_image_recolor(&style, LV_STATE_DEFAULT, LV_COLOR_BLUE);
lv_style_set_image_recolor_opa(&style, LV_STATE_DEFAULT, LV_OPA_50);

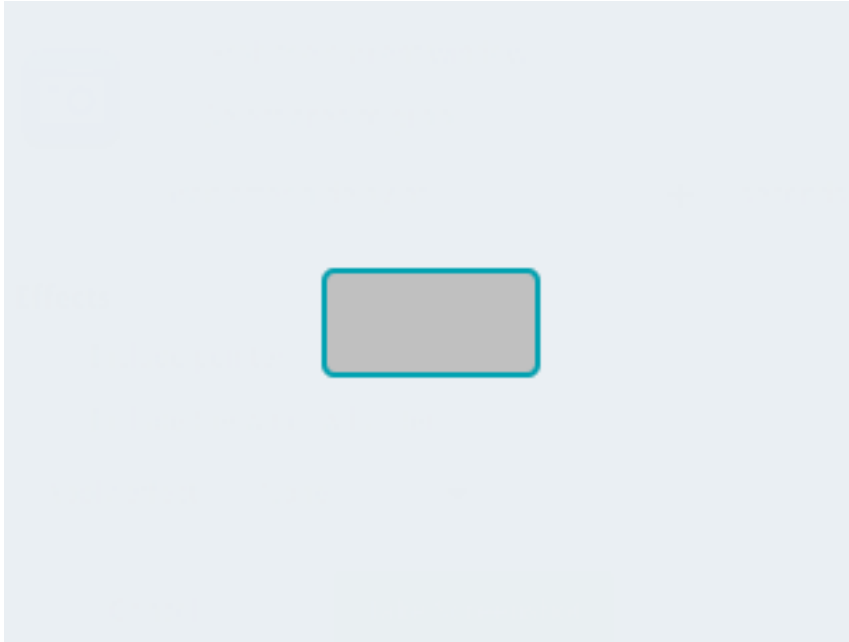
#if LV_USE_IMG
/*Create an object with the new style*/
lv_obj_t * obj = lv_img_create(lv_scr_act(), NULL);
lv_obj_add_style(obj, LV_IMG_PART_MAIN, &style);
LV_IMG_DECLARE(img_cogwheel_argb);
lv_img_set_src(obj, &img_cogwheel_argb);
lv_obj_align(obj, NULL, LV_ALIGN_CENTER, 0, 0);
#endif
}

```

Transition properties

Properties to describe state change animations.

- **transition_time** (`lv_style_int_t`): Time of the transition. Default value: 0.
- **transition_delay** (`lv_style_int_t`): Delay before the transition. Default value: 0.
- **transition_prop_1** (property name): A property on which transition should be applied. Use the property name with upper case with `LV_STYLE_` prefix, e.g. `LV_STYLE_BG_COLOR`. Default value: 0 (none).
- **transition_prop_2** (property name): Same as *transition_1* just for another property. Default value: 0 (none).
- **transition_prop_3** (property name): Same as *transition_1* just for another property. Default value: 0 (none).
- **transition_prop_4** (property name): Same as *transition_1* just for another property. Default value: 0 (none).
- **transition_prop_5** (property name): Same as *transition_1* just for another property. Default value: 0 (none).
- **transition_prop_6** (property name): Same as *transition_1* just for another property. Default value: 0 (none).
- **transition_path** (`lv_anim_path_t`): An animation path for the transition. (Needs to be static or global variable because only its pointer is saved). Default value: `lv_anim_path_def` (linear path).



```
#include "../../lv_examples.h"

/**
 * Using the transitions style properties
 */
void lv_ex_style_10(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, LV_STATE_DEFAULT, 5);
    lv_style_set_bg_opa(&style, LV_STATE_DEFAULT, LV_OPA_COVER);
    lv_style_set_bg_color(&style, LV_STATE_DEFAULT, LV_COLOR_SILVER);

    /*Set different background color in pressed state*/
    lv_style_set_bg_color(&style, LV_STATE_PRESSED, LV_COLOR_GRAY);

    /*Set different transition time in default and pressed state
     *fast press, slower revert to default*/
    lv_style_set_transition_time(&style, LV_STATE_DEFAULT, 500);
    lv_style_set_transition_time(&style, LV_STATE_PRESSED, 200);

    /*Small delay to make transition more visible*/
    lv_style_set_transition_delay(&style, LV_STATE_DEFAULT, 100);

    /*Add `bg_color` to transitioned properties*/
    lv_style_set_transition_prop_1(&style, LV_STATE_DEFAULT, LV_STYLE_BG_COLOR);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_scr_act(), NULL);
    lv_obj_add_style(obj, LV_OBJ_PART_MAIN, &style);
    lv_obj_align(obj, NULL, LV_ALIGN_CENTER, 0, 0);
}
```

Scale properties

Auxiliary properties for scale-like elements. Scales have a normal and end region. As the name implies the end region is the end of the scale where can be critical values or inactive values. The normal region is before the end region. Both regions could have different properties.

- **scale_grad_color** (lv_color_t): In normal region make gradient to this color on the scale lines. Default value: LV_COLOR_BLACK.
- **scale_end_color** (lv_color_t): Color of the scale lines in the end region. Default value: LV_COLOR_BLACK.
- **scale_width** (lv_style_int_t): Width of the scale. Default value: LV_DPI / 8. Default value: LV_DPI / 8.
- **scale_border_width** (lv_style_int_t): Width of a border drawn on the outer side of the scale in the normal region. Default value: 0.
- **scale_end_border_width** (lv_style_int_t): Width of a border drawn on the outer side of the scale in the end region. Default value: 0.
- **scale_end_line_width** (lv_style_int_t): Width of a scale lines in the end region. Default value: 0.



```
#include "../lv_examples.h"

/**
 * Using the scale style properties
 */
void lv_ex_style_11(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, LV_STATE_DEFAULT, 5);
    lv_style_set_bg_opa(&style, LV_STATE_DEFAULT, LV_OPA_COVER);
```

(continues on next page)

(continued from previous page)

```

lv_style_set_bg_color(&style, LV_STATE_DEFAULT, LV_COLOR_SILVER);

/*Set some paddings*/
lv_style_set_pad_inner(&style, LV_STATE_DEFAULT, 20);
lv_style_set_pad_top(&style, LV_STATE_DEFAULT, 20);
lv_style_set_pad_left(&style, LV_STATE_DEFAULT, 5);
lv_style_set_pad_right(&style, LV_STATE_DEFAULT, 5);

lv_style_set_scale_end_color(&style, LV_STATE_DEFAULT, LV_COLOR_RED);
lv_style_set_line_color(&style, LV_STATE_DEFAULT, LV_COLOR_WHITE);
lv_style_set_scale_grad_color(&style, LV_STATE_DEFAULT, LV_COLOR_BLUE);
lv_style_set_line_width(&style, LV_STATE_DEFAULT, 2);
lv_style_set_scale_end_line_width(&style, LV_STATE_DEFAULT, 4);
lv_style_set_scale_end_border_width(&style, LV_STATE_DEFAULT, 4);

/*Gauge has a needle but for simplicity its style is not initialized here*/
#if LV_USE_GAUGE
/*Create an object with the new style*/
lv_obj_t * obj = lv_gauge_create(lv_scr_act(), NULL);
lv_obj_add_style(obj, LV_GAUGE_PART_MAIN, &style);
lv_obj_align(obj, NULL, LV_ALIGN_CENTER, 0, 0);
#endif
}

```

In the documentation of the widgets you will see sentences like "The widget use the typical background properties". The "typical background" properties are:

- Background
- Border
- Outline
- Shadow
- Pattern
- Value

4.4.10 Themes

Themes are a collection of styles. There is always an active theme whose styles are automatically applied when an object is created. It gives a default appearance to UI which can be modified by adding further styles.

The default theme is set in `lv_conf.h` with `LV_THEME_...` defines. Every theme has the following properties

- primary color
- secondary color
- small font
- normal font
- subtitle font
- title font

- flags (specific to the given theme)

It up to the theme how to use these properties.

There are 3 built-in themes:

- empty: no default styles are added
- material: an impressive, modern theme - mono: simple black and white theme for monochrome displays
- template: a very simple theme which can be copied to create a custom theme

Extending themes

Built-in themes can be extended by custom theme. If a custom theme is created a "base theme" can be selected. The base theme's styles will be added before the custom theme. Any number of themes can be chained this way. E.g. material theme -> custom theme -> dark theme.

Here is an example about how to create a custom theme based on the currently active built-in theme.

```
/*Get the current theme (e.g. material). It will be the base of the custom theme.*/
lv_theme_t * base_theme = lv_theme_get_act();

/*Initialize a custom theme*/
static lv_theme_t custom_theme;                                /*Declare a theme*/
lv_theme_copy(&custom_theme, base_theme);                       /*Initialize the custom theme
↳ from the base theme*/
lv_theme_set_apply_cb(&custom_theme, custom_apply_cb);          /*Set a custom theme apply
↳ callback*/
lv_theme_set_base(custom_theme, base_theme);                    /*Set the base theme of the
↳ custom theme*/

/*Initialize styles for the new theme*/
static lv_style_t style1;
lv_style_init(&style1);
lv_style_set_bg_color(&style1, LV_STATE_DEFAULT, custom_theme.color_primary);

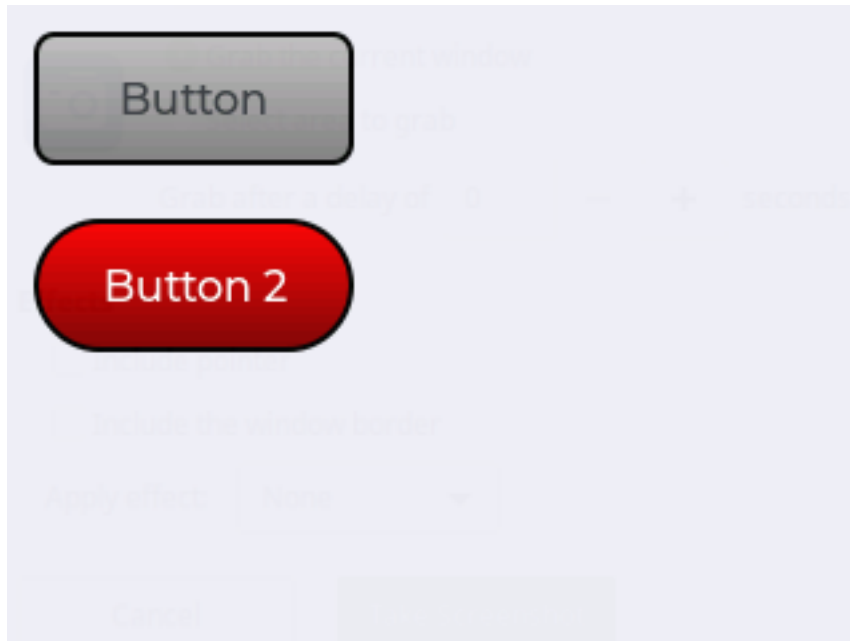
...

/*Add a custom apply callback*/
static void custom_apply_cb(lv_theme_t * th, lv_obj_t * obj, lv_theme_style_t name)
{
    lv_style_list_t * list;

    switch(name) {
        case LV_THEME_BTN:
            list = lv_obj_get_style_list(obj, LV_BTN_PART_MAIN);
            _lv_style_list_add_style(list, &my_style);
            break;
    }
}
```

4.4.11 Example

Styling a button



```
#include "../../lv_examples.h"

/**
 * Create styles from scratch for buttons.
 */
void lv_ex_get_started_2(void)
{
    static lv_style_t style_btn;
    static lv_style_t style_btn_red;

    /*Create a simple button style*/
    lv_style_init(&style_btn);
    lv_style_set_radius(&style_btn, LV_STATE_DEFAULT, 10);
    lv_style_set_bg_opa(&style_btn, LV_STATE_DEFAULT, LV_OPA_COVER);
    lv_style_set_bg_color(&style_btn, LV_STATE_DEFAULT, LV_COLOR_SILVER);
    lv_style_set_bg_grad_color(&style_btn, LV_STATE_DEFAULT, LV_COLOR_GRAY);
    lv_style_set_bg_grad_dir(&style_btn, LV_STATE_DEFAULT, LV_GRAD_DIR_VER);

    /*Swap the colors in pressed state*/
    lv_style_set_bg_color(&style_btn, LV_STATE_PRESSED, LV_COLOR_GRAY);
    lv_style_set_bg_grad_color(&style_btn, LV_STATE_PRESSED, LV_COLOR_SILVER);

    /*Add a border*/
    lv_style_set_border_color(&style_btn, LV_STATE_DEFAULT, LV_COLOR_WHITE);
    lv_style_set_border_opa(&style_btn, LV_STATE_DEFAULT, LV_OPA_70);
    lv_style_set_border_width(&style_btn, LV_STATE_DEFAULT, 2);

    /*Different border color in focused state*/
    lv_style_set_border_color(&style_btn, LV_STATE_FOCUSED, LV_COLOR_BLUE);
}
```

(continues on next page)

(continued from previous page)

```

    lv_style_set_border_color(&style_btn, LV_STATE_FOCUSED | LV_STATE_PRESSED, LV_
↪COLOR_NAVY);

    /*Set the text style*/
    lv_style_set_text_color(&style_btn, LV_STATE_DEFAULT, LV_COLOR_WHITE);

    /*Make the button smaller when pressed*/
    lv_style_set_transform_height(&style_btn, LV_STATE_PRESSED, -5);
    lv_style_set_transform_width(&style_btn, LV_STATE_PRESSED, -10);
    #if LV_USE_ANIMATION
    /*Add a transition to the size change*/
    static lv_anim_path_t path;
    lv_anim_path_init(&path);
    lv_anim_path_set_cb(&path, lv_anim_path_overshoot);

    lv_style_set_transition_prop_1(&style_btn, LV_STATE_DEFAULT, LV_STYLE_TRANSFORM_
↪HEIGHT);
    lv_style_set_transition_prop_2(&style_btn, LV_STATE_DEFAULT, LV_STYLE_TRANSFORM_
↪WIDTH);
    lv_style_set_transition_time(&style_btn, LV_STATE_DEFAULT, 300);
    lv_style_set_transition_path(&style_btn, LV_STATE_DEFAULT, &path);
    #endif

    /*Create a red style. Change only some colors.*/
    lv_style_init(&style_btn_red);
    lv_style_set_bg_color(&style_btn_red, LV_STATE_DEFAULT, LV_COLOR_RED);
    lv_style_set_bg_grad_color(&style_btn_red, LV_STATE_DEFAULT, LV_COLOR_MAROON);
    lv_style_set_bg_color(&style_btn_red, LV_STATE_PRESSED, LV_COLOR_MAROON);
    lv_style_set_bg_grad_color(&style_btn_red, LV_STATE_PRESSED, LV_COLOR_RED);
    lv_style_set_text_color(&style_btn_red, LV_STATE_DEFAULT, LV_COLOR_WHITE);
    #if LV_USE_BTN
    /*Create buttons and use the new styles*/
    lv_obj_t * btn = lv_btn_create(lv_scr_act(), NULL);          /*Add a button the
↪current screen*/
    lv_obj_set_pos(btn, 10, 10);                                  /*Set its position*/
    lv_obj_set_size(btn, 120, 50);                                /*Set its size*/
    lv_obj_reset_style_list(btn, LV_BTN_PART_MAIN);               /*Remove the styles
↪coming from the theme*/
    lv_obj_add_style(btn, LV_BTN_PART_MAIN, &style_btn);

    lv_obj_t * label = lv_label_create(btn, NULL);                /*Add a label to the
↪button*/
    lv_label_set_text(label, "Button");                            /*Set the labels text*/

    /*Create a new button*/
    lv_obj_t * btn2 = lv_btn_create(lv_scr_act(), btn);
    lv_obj_set_pos(btn2, 10, 80);
    lv_obj_set_size(btn2, 120, 50);                               /*Set its size*/
    lv_obj_reset_style_list(btn2, LV_BTN_PART_MAIN);              /*Remove the styles
↪coming from the theme*/
    lv_obj_add_style(btn2, LV_BTN_PART_MAIN, &style_btn);
    lv_obj_add_style(btn2, LV_BTN_PART_MAIN, &style_btn_red);     /*Add the red style
↪on top of the current */
    lv_obj_set_style_local_radius(btn2, LV_BTN_PART_MAIN, LV_STATE_DEFAULT, LV_RADIUS_
↪CIRCLE); /*Add a local style*/

```

(continues on next page)

(continued from previous page)

```

    label = lv_label_create(btn2, NULL);          /*Add a label to the button*/
    lv_label_set_text(label, "Button 2");         /*Set the labels text*/
#endif
}

```

4.5 Input devices

An input device usually means:

- Pointer-like input device like touchpad or mouse
- Keypads like a normal keyboard or simple numeric keypad
- Encoders with left/right turn and push options
- External hardware buttons which are assigned to specific points on the screen

Important: Before reading further, please read the [Porting](/porting/indev) section of Input devices

4.5.1 Pointers

Pointer input devices can have a cursor. (typically for mouses)

```

...
lv_indev_t * mouse_indev = lv_indev_drv_register(&indev_drv);

LV_IMG_DECLARE(mouse_cursor_icon);           /*Declare the image file.
↪*/
lv_obj_t * cursor_obj = lv_img_create(lv_scr_act(), NULL); /*Create an image object
↪for the cursor */
lv_img_set_src(cursor_obj, &mouse_cursor_icon);           /*Set the image source*/
lv_indev_set_cursor(mouse_indev, cursor_obj);             /*Connect the image
↪object to the driver*/

```

Note that the cursor object should have `lv_obj_set_click(cursor_obj, false)`. For images, *clicking* is disabled by default.

4.5.2 Keypad and encoder

You can fully control the user interface without touchpad or mouse using a keypad or encoder(s). It works similar to the *TAB* key on the PC to select the element in an application or a web page.

Groups

The objects, you want to control with keypad or encoder, needs to be added to a *Group*. In every group, there is exactly one focused object which receives the pressed keys or the encoder actions. For example, if a *Text area* is focused and you press some letter on a keyboard, the keys will be sent and inserted into the text area. Similarly, if a *Slider* is focused and you press the left or right arrows, the slider's value will be changed.

You need to associate an input device with a group. An input device can send the keys to only one group but, a group can receive data from more than one input device too.

To create a group use `lv_group_t * g = lv_group_create()` and to add an object to the group use `lv_group_add_obj(g, obj)`.

To associate a group with an input device use `lv_indev_set_group(indev, g)`, where `indev` is the return value of `lv_indev_drv_register()`

Keys

There are some predefined keys which have special meaning:

- **LV_KEY_NEXT** Focus on the next object
- **LV_KEY_PREV** Focus on the previous object
- **LV_KEY_ENTER** Triggers **LV_EVENT_PRESSED/CLICKED/LONG_PRESSED** etc. events
- **LV_KEY_UP** Increase value or move upwards
- **LV_KEY_DOWN** Decrease value or move downwards
- **LV_KEY_RIGHT** Increase value or move the the right
- **LV_KEY_LEFT** Decrease value or move the the left
- **LV_KEY_ESC** Close or exit (E.g. close a *Drop down list*)
- **LV_KEY_DEL** Delete (E.g. a character on the right in a *Text area*)
- **LV_KEY_BACKSPACE** Delete a character on the left (E.g. in a *Text area*)
- **LV_KEY_HOME** Go to the beginning/top (E.g. in a *Text area*)
- **LV_KEY_END** Go to the end (E.g. in a *Text area*)

The most important special keys are **LV_KEY_NEXT/PREV**, **LV_KEY_ENTER** and **LV_KEY_UP/DOWN/LEFT/RIGHT**. In your `read_cb` function, you should translate some of your keys to these special keys to navigate in the group and interact with the selected object.

Usually, it's enough to use only **LV_KEY_LEFT/RIGHT** because most of the objects can be fully controlled with them.

With an encoder, you should use only **LV_KEY_LEFT**, **LV_KEY_RIGHT**, and **LV_KEY_ENTER**.

Edit and navigate mode

Since a keypad has plenty of keys, it's easy to navigate between the objects and edit them using the keypad. But, the encoders have a limited number of "keys" hence, it is difficult to navigate using the default options. *Navigate* and *Edit* are created to avoid this problem with the encoders.

In *Navigate* mode, the encoders `LV_KEY_LEFT/RIGHT` is translated to `LV_KEY_NEXT/PREV`. Therefore the next or previous object will be selected by turning the encoder. Pressing `LV_KEY_ENTER` will change to *Edit* mode.

In *Edit* mode, `LV_KEY_NEXT/PREV` is usually used to edit the object. Depending on the object's type, a short or long press of `LV_KEY_ENTER` changes back to *Navigate* mode. Usually, an object which can not be pressed (like a *Slider*) leaves *Edit* mode on short click. But with objects where short click has meaning (e.g. *Button*), a long press is required.

Styling

If an object is focused either by clicking it via touchpad, or focused via an encoder or keypad it goes to `LV_STATE_FOCUSED`. Hence focused styles will be applied on it.

If the object goes to edit mode it goes to `LV_STATE_FOCUSED | LV_STATE_EDITED` state so these style properties will be shown.

For a more detailed description read the [Style](#) section.

4.6 Displays

Important: The basic concept of *display* in LVGL is explained in the [\[Porting\]\(/porting/display\)](#) section. So before reading further, please read the [\[Porting\]\(/porting/display\)](#) section first.

4.6.1 Multiple display support

In LVGL, you can have multiple displays, each with their own driver and objects. The only limitation is that every display needs to have same color depth (as defined in `LV_COLOR_DEPTH`). If the displays are different in this regard the rendered image can be converted to the correct format in the drivers `flush_cb`.

Creating more displays is easy: just initialize more display buffers and register another driver for every display. When you create the UI, use `lv_disp_set_default(disp)` to tell the library on which display to create objects.

Why would you want multi-display support? Here are some examples:

- Have a "normal" TFT display with local UI and create "virtual" screens on VNC on demand. (You need to add your VNC driver).
- Have a large TFT display and a small monochrome display.
- Have some smaller and simple displays in a large instrument or technology.
- Have two large TFT displays: one for a customer and one for the shop assistant.

Using only one display

Using more displays can be useful, but in most cases, it's not required. Therefore, the whole concept of multi-display is completely hidden if you register only one display. By default, the lastly created (the only one) display is used as default.

`lv_scr_act()`, `lv_scr_load(scr)`, `lv_layer_top()`, `lv_layer_sys()`, `LV_HOR_RES` and `LV_VER_RES` are always applied on the lastly created (default) screen. If you pass `NULL` as `disp` parameter to display related function, usually the default display will be used. E.g. `lv_disp_trig_activity(NULL)` will trigger a user activity on the default screen. (See below in *In-activity*).

Mirror display

To mirror the image of the display to another display, you don't need to use the multi-display support. Just transfer the buffer received in `drv.flush_cb` to another display too.

Split image

You can create a larger display from smaller ones. You can create it as below:

1. Set the resolution of the displays to the large display's resolution.
2. In `drv.flush_cb`, truncate and modify the `area` parameter for each display.
3. Send the buffer's content to each display with the truncated area.

4.6.2 Screens

Every display has each set of [Screens](#) and the object on the screens.

Be sure not to confuse displays and screens:

- **Displays** are the physical hardware drawing the pixels.
- **Screens** are the high-level root objects associated with a particular display. One display can have multiple screens associated with it, but not vice versa.

Screens can be considered the highest level containers which have no parent. The screen's size is always equal to its display and size their position is (0;0). Therefore, the screens coordinates can't be changed, i.e. `lv_obj_set_pos()`, `lv_obj_set_size()` or similar functions can't be used on screens.

A screen can be created from any object type but, the two most typical types are the *Base object* and the *Image* (to create a wallpaper).

To create a screen, use `lv_obj_t * scr = lv_<type>_create(NULL, copy)`. `copy` can be an other screen to copy it.

To load a screen, use `lv_scr_load(scr)`. To get the active screen, use `lv_scr_act()`. These functions works on the default display. If you want to specify which display to work on, use `lv_disp_get_scr_act(disp)` and `lv_disp_load_scr(disp, scr)`. Screen can be loaded with animations too. Read more [here](#).

Screens can be deleted with `lv_obj_del(scr)`, but ensure that you do not delete the currently loaded screen.

Transparent screens

Usually, the opacity of the screen is `LV_OPA_COVER` to provide a solid background for its children. If it's not the case (opacity < 100%) the display's background color or image will be visible. See the *Display background* section for more details. If the display's background opacity is also not `LV_OPA_COVER` LVGL has no solid background to draw.

This configuration (transparent screen and display) could be used to create for example OSD menus where a video is played to lower layer, and menu is created on an upper layer.

To handle transparent displays special (slower) color mixing algorithms needs to be used by LVGL so this feature needs to be enabled with `LV_COLOR_SCREEN_TRANSP` in `lv_conf.h`. As this mode operates on the Alpha channel of the pixels `LV_COLOR_DEPTH = 32` is also required. The Alpha channel of 32-bit colors will be 0 where there are no objects and will be 255 where there are solid objects.

In summary, to enable transparent screen and displays to create OSD menu-like UIs:

- Enable `LV_COLOR_SCREEN_TRANSP` in `lv_conf.h`
- Be sure to use `LV_COLOR_DEPTH 32`
- Set the screen's opacity to `LV_OPA_TRANSP` e.g. with `lv_obj_set_style_local_bg_opa(lv_scr_act(), LV_OBMASK_PART_MAIN, LV_STATE_DEFAULT, LV_OPA_TRANSP)`
- Set the display opacity to `LV_OPA_TRANSP` with `lv_disp_set_bg_opa(NULL, LV_OPA_TRANSP);`

4.6.3 Features of displays

Inactivity

The user's inactivity is measured on each display. Every use of an *Input device* (if associated with the display) counts as an activity. To get time elapsed since the last activity, use `lv_disp_get_inactive_time(displ)`. If `NULL` is passed, the overall smallest inactivity time will be returned from all displays (**not the default display**).

You can manually trigger an activity using `lv_disp_trig_activity(displ)`. If `displ` is `NULL`, the default screen will be used (**and not all displays**).

Background

Every display has background color, a background image and background opacity properties. They become visible when the current screen is transparent or not positioned to cover the whole display.

Background color is a simple color to fill the display. It can be adjusted with `lv_disp_set_bg_color(displ, color);`

Background image is path to file or pointer to an `lv_img_dsc_t` variable (converted image) to be used as wallpaper. It can be set with `lv_disp_set_bg_color(displ, &my_img);` If the background image is set (not `NULL`) the background won't be filled with `bg_color`.

The opacity of the background color or image can be adjusted with `lv_disp_set_bg_opa(displ, opa)`.

The `displ` parameter of these functions can be `NULL` to refer it to the default display.

4.6.4 Colors

The color module handles all color-related functions like changing color depth, creating colors from hex code, converting between color depths, mixing colors, etc.

The following variable types are defined by the color module:

- **lv_color1_t** Store monochrome color. For compatibility, it also has R, G, B fields but they are always the same value (1 byte)
- **lv_color8_t** A structure to store R (3 bit),G (3 bit),B (2 bit) components for 8-bit colors (1 byte)
- **lv_color16_t** A structure to store R (5 bit),G (6 bit),B (5 bit) components for 16-bit colors (2 byte)
- **lv_color32_t** A structure to store R (8 bit),G (8 bit), B (8 bit) components for 24-bit colors (4 byte)
- **lv_color_t** Equal to **lv_color1/8/16/24_t** according to color depth settings
- **lv_color_int_t** **uint8_t**, **uint16_t** or **uint32_t** according to color depth setting. Used to build color arrays from plain numbers.
- **lv_opa_t** A simple **uint8_t** type to describe opacity.

The **lv_color_t**, **lv_color1_t**, **lv_color8_t**, **lv_color16_t** and **lv_color32_t** types have got four fields:

- **ch.red** red channel
- **ch.green** green channel
- **ch.blue** blue channel
- **full** red + green + blue as one number

You can set the current color depth in *lv_conf.h*, by setting the **LV_COLOR_DEPTH** define to 1 (monochrome), 8, 16 or 32.

Convert color

You can convert a color from the current color depth to another. The converter functions return with a number, so you have to use the **full** field:

```
lv_color_t c;
c.red   = 0x38;
c.green = 0x70;
c.blue  = 0xCC;

lv_color1_t c1;
c1.full = lv_color_to1(c);           /*Return 1 for light colors, 0 for dark colors*/

lv_color8_t c8;
c8.full = lv_color_to8(c);          /*Give a 8 bit number with the converted color*/

lv_color16_t c16;
c16.full = lv_color_to16(c); /*Give a 16 bit number with the converted color*/

lv_color32_t c32;
c32.full = lv_color_to32(c);        /*Give a 32 bit number with the converted color*/
```

Swap 16 colors

You may set `LV_COLOR_16_SWAP` in `lv_conf.h` to swap the bytes of *RGB565* colors. It's useful if you send the 16-bit colors via a byte-oriented interface like SPI.

As 16-bit numbers are stored in Little Endian format (lower byte on the lower address), the interface will send the lower byte first. However, displays usually need the higher byte first. A mismatch in the byte order will result in highly distorted colors.

Create and mix colors

You can create colors with the current color depth using the `LV_COLOR_MAKE` macro. It takes 3 arguments (red, green, blue) as 8-bit numbers. For example to create light red color: `my_color = COLOR_MAKE(0xFF, 0x80, 0x80)`.

Colors can be created from HEX codes too: `my_color = lv_color_hex(0x288ACF)` or `my_color = lv_folor_hex3(0x28C)`.

Mixing two colors is possible with `mixed_color = lv_color_mix(color1, color2, ratio)`. Ratio can be 0..255. 0 results fully color2, 255 result fully color1.

Colors can be created with from HSV space too using `lv_color_hsv_to_rgb(hue, saturation, value)`. `hue` should be in 0..360 range, `saturation` and `value` in 0..100 range.

Opacity

To describe opacity the `lv_opa_t` type is created as a wrapper to `uint8_t`. Some defines are also introduced:

- `LV_OPA_TRANSP` Value: 0, means the opacity makes the color completely transparent
- `LV_OPA_10` Value: 25, means the color covers only a little
- `LV_OPA_20 ... OPA_80` come logically
- `LV_OPA_90` Value: 229, means the color near completely covers
- `LV_OPA_COVER` Value: 255, means the color completely covers

You can also use the `LV_OPA_*` defines in `lv_color_mix()` as a *ratio*.

Built-in colors

The color module defines the most basic colors such as:

- `LV_COLOR_WHITE`
-  `LV_COLOR_BLACK`
-  `LV_COLOR_GRAY`
-  `LV_COLOR_SILVER`
-  `LV_COLOR_RED`
-  `LV_COLOR_MAROON`
-  `LV_COLOR_LIME`

-  LV_COLOR_GREEN
-  LV_COLOR_OLIVE
-  LV_COLOR_BLUE
-  LV_COLOR_NAVY
-  LV_COLOR_TEAL
-  LV_COLOR_CYAN
-  LV_COLOR_AQUA
-  LV_COLOR_PURPLE
-  LV_COLOR_MAGENTA
-  LV_COLOR_ORANGE
-  LV_COLOR_YELLOW

as well as LV_COLOR_WHITE (fully white).

4.7 Fonts

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

```
lv_style_set_text_font(&my_style, LV_STATE_DEFAULT, &lv_font_montserrat_28); /*Set a ↵
↪larger font*/
```

The fonts have a **bpp (bits 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. For example, *bpp* = 4 makes the font nearly 4 times greater compared to *bpp* = 1.

4.7.1 Unicode support

LVGL supports **UTF-8** encoded Unicode characters. Your editor needs to be configured to save your code/text as UTF-8 (usually this the default) and be sure that, `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 * label1 = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label1, 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_MONTERRAT_12` 12 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_14` 14 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_16` 16 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_18` 18 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_20` 20 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_22` 22 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_24` 24 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_26` 26 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_28` 28 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_30` 30 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_32` 32 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_34` 34 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_36` 36 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_38` 38 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_40` 40 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_42` 42 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_44` 44 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_46` 46 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_48` 48 px ASCII + built-in symbol
- `LV_FONT_MONTERRAT_12_SUBPX` 12 px font with *subpixel rendering*
- `LV_FONT_MONTERRAT_28_COMPRESSED` 28 px *compressed font* with 3 bpp
- `LV_FONT_DEJAVU_16_PERSIAN_HEBREW` 16 px Hebrew, Arabic, Persian letters and all their forms
- `LV_FONT_SIMSUN_16_CJK` 16 px 1000 most common CJK radicals
- `LV_FONT_UNSCII_8` 8 px pixel perfect font

The built-in fonts are **global variables** with names like `lv_font_montserrat_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 [Montserrat](#) 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_WARNING
	LV_SYMBOL_VIDEO		LV_SYMBOL_SHUFFLE
	LV_SYMBOL_LIST		LV_SYMBOL_UP
	LV_SYMBOL_OK		LV_SYMBOL_DOWN
	LV_SYMBOL_CLOSE		LV_SYMBOL_LOOP
	LV_SYMBOL_POWER		LV_SYMBOL_DIRECTORY
	LV_SYMBOL_SETTINGS		LV_SYMBOL_UPLOAD
	LV_SYMBOL_TRASH		LV_SYMBOL_CALL
	LV_SYMBOL_HOME		LV_SYMBOL_CUT
	LV_SYMBOL_DOWNLOAD		LV_SYMBOL_COPY
	LV_SYMBOL_DRIVE		LV_SYMBOL_SAVE
	LV_SYMBOL_REFRESH		LV_SYMBOL_CHARGE
	LV_SYMBOL_MUTE		LV_SYMBOL_PASTE
	LV_SYMBOL_VOLUME_MID		LV_SYMBOL_BELL
	LV_SYMBOL_VOLUME_MAX		LV_SYMBOL_KEYBOARD
	LV_SYMBOL_IMAGE		LV_SYMBOL_GPS
	LV_SYMBOL_EDIT		LV_SYMBOL_FILE
	LV_SYMBOL_PREV		LV_SYMBOL_WIFI
	LV_SYMBOL_PLAY		LV_SYMBOL_BATTERY_FULL
	LV_SYMBOL_PAUSE		LV_SYMBOL_BATTERY_3
	LV_SYMBOL_STOP		LV_SYMBOL_BATTERY_2
	LV_SYMBOL_NEXT		LV_SYMBOL_BATTERY_1
	LV_SYMBOL_EJECT		LV_SYMBOL_BATTERY_EMPTY
	LV_SYMBOL_LEFT		LV_SYMBOL_USB
	LV_SYMBOL_RIGHT		LV_SYMBOL_BLUETOOTH
	LV_SYMBOL_PLUS		LV_SYMBOL_BACKSPACE
	LV_SYMBOL_MINUS		LV_SYMBOL_SD_CARD
	LV_SYMBOL_EYE_OPEN		LV_SYMBOL_NEW_LINE
	LV_SYMBOL_EYE_CLOSE		

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 Special features

Bidirectional support

Most of the languages use Left-to-Right (LTR for short) writing direction, however some languages (such as Hebrew, Persian or Arabic) uses Right-to-Left (RTL for short) direction.

LVGL not only supports RTL texts but supports mixed (a.k.a. bidirectional, BiDi) text rendering too. Some examples:

The names of these states in Arabic
are مصر, البحرين and الكويت respectively.

The title is مفتاح معايير الويب! in Arabic.

The BiDi support can be enabled by `LV_USE_BIDI` in `lv_conf.h`

All texts have a base direction (LTR or RTL) which determines some rendering rules and the default alignment of the text (Left or Right). However, in LVGL, base direction is applied not only for labels. It's a general property which can be set for every object. If unset then it will be inherited from the parent. So it's enough to set the base direction of the screen and every object will inherit it.

The default base direction of screen can be set by `LV_BIDI_BASE_DIR_DEF` in `lv_conf.h` and other objects inherit the base direction from their parent.

To set an object's base direction use `lv_obj_set_base_dir(obj, base_dir)`. The possible base direction are:

- `LV_BIDI_DIR_LTR`: Left to Right base direction
- `LV_BIDI_DIR_RTL`: Right to Left base direction
- `LV_BIDI_DIR_AUTO`: Auto detect base direction
- `LV_BIDI_DIR_INHERIT`: Inherit the base direction from the parent (default for non-screen objects)

This list summarizes the effect of RTL base direction on objects:

- Create objects by default on the right
- `lv_tabview`: displays tabs from right to left
- `lv_checkbox`: Show the box on the right
- `lv_btnmatrix`: Show buttons from right to left
- `lv_list`: Show the icon on the right
- `lv_dropdown`: Align the options to the right
- The texts in `lv_table`, `lv_btnmatrix`, `lv_keyboard`, `lv_tabview`, `lv_dropdown`, `lv_roller` are "BiDi processed" to be displayed correctly

Arabic and Persian support

There are some special rules to display Arabic and Persian characters: the *form* of the character depends on their position in the text. A different form of the same letter needs to be used if it isolated, start, middle or end position. Besides these some conjunction rules also should be taken into account.

LVGL supports to apply these rules if `LV_USE_ARABIC_PERSIAN_CHARS` is enabled.

However, there some limitations:

- Only displaying texts is supported (e.g. on labels), text inputs (e.g. text area) doesn't support this feature
- Static text (i.e. `const`) are not processed. E.g. texts set by `lv_label_set_text()` will "Arabic processed" but `lv_label_set_text_static()` won't.
- Text get functions (e.g. `lv_label_get_text()`) will return the processed text.

Subpixel rendering

Subpixel rendering means to triple the horizontal resolution by rendering on Red, Green and Blue channel instead of pixel level. It takes advantage of the position of physical color channels of each pixel. It results in higher quality letter anti-aliasing. Learn more [here](#).

Subpixel rendering requires to generate the fonts with special settings:

- In the online converter tick the **Subpixel** box
- In the command line tool use `--lcd` flag. Note that the generated font needs about 3 times more memory.

Subpixel rendering works only if the color channels of the pixels have a horizontal layout. That is the R, G, B channels are next each other and not above each other. The order of color channels also needs to match with the library settings. By default the LVGL assumes **RGB** order, however it can be swapped by setting `LV_SUBPX_BGR 1` in `lv_conf.h`.

Compress fonts

The bitmaps of the fonts can be compressed by

- ticking the **Compressed** check box in the online converter
- not passing `--no-compress` flag to the offline converter (applies compression by default)

The compression is more effective with larger fonts and higher bpp. However, it's about 30% slower to render the compressed fonts. Therefore it's recommended to compress only the largest fonts of user interface, because

- they need the most memory
- they can be compressed better
- and probably they are used less frequently then the medium sized fonts. (so performance cost is smaller)

4.7.4 Add new font

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

1. The simplest method is to use the [Online font converter](#). Just set the parameters, click the *Convert* button, copy the font to your project and use it. **Be sure to carefully read the steps provided on that site or you will get an error while converting.**
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 the fonts globally available (like the builtin fonts), add them to `LV_FONT_CUSTOM_DECLARE` in `lv_conf.h`.

4.7.5 Add new symbols

The built-in symbols are created from [FontAwesome](#) font.

1. Search symbol on <https://fontawesome.com>. For example the [USB symbol](#). Copy it's Unicode ID which is `0xf287` in this case.
2. Open the [Online font converter](#). Add Add [FontAwesome.woff](#). .
3. Set the parameters such as Name, Size, BPP. You'll use this name to declare and use the font in your code.
4. Add the Unicode ID of the symbol to the range field. E.g. `0xf287` for the USB symbol. More symbols can be enumerated with `,`.
5. Convert the font and copy it to your project. Make sure to compile the `.c` file of your font.
6. Declare the font using `extern lv_font_t my_font_name;` or simply `LV_FONT_DECLARE(my_font_name);`.

Using the symbol

1. 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`.
2. Create a `define` from the UTF8 values: `#define MY_USB_SYMBOL "\xEF\x8A\x87"`
3. Create a label and set the text. Eg. `lv_label_set_text(label, MY_USB_SYMBOL)`

Note - `lv_label_set_text(label, MY_USB_SYMBOL)` searches for this symbol in the font defined in `style.text.font` properties. To use the symbol you may need to change it. Eg `style.text.font = my_font_name`

4.7.6 Load font in run-time

`lv_font_load` can be used to load a font from a file. The font to load needs to have a special binary format. (Not TTF or WOFF). Use `lv_font_conv` with `--format bin` option to generate an LVGL compatible font file.

Note that to load a font *LVGL's filesystem* needs to be enabled and a driver needs to be added.

Example

```
lv_font_t * my_font;
my_font = lv_font_load(X/path/to/my_font.bin);

/*Use the font*/

/*Free the font if not required anymore*/
lv_font_free(my_font);
```

4.7.7 Add a new font engine

LVGL's font interface is designed to be very flexible. You don't need to use LVGL'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 needs them.

A raedy to use FreeType can be found in `lv_freetype` repository.

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*/

    /* Store the result.
     * For example ...
```

(continues on next page)

(continued from previous page)

```

    */
    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 offset of the bitmap measured from the as line*/
    dsc_out->bpp = 2;         /*Bits per pixel: 1/2/4/8*/

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

/* Get the bitmap 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

An image can be a file or variable which stores the bitmap itself and some metadata.

4.8.1 Store images

You can store images in two places

- as a variable in the internal memory (RAM or ROM)
- as a file

Variables

The images stored internally in a variable is composed mainly of an `lv_img_dsc_t` structure with the following fields:

- **header**
 - *cf* Color format. See *below*
 - *w* width in pixels (≤ 2048)
 - *h* height in pixels (≤ 2048)
 - *always zero* 3 bits which need to be always zero
 - *reserved* reserved for future use
- **data** pointer to an array where the image itself is stored
- **data_size** length of **data** in bytes

These are usually stored within a project as C files. They are linked into the resulting executable like any other constant data.

Files

To deal with files you need to add a *Drive* to LVGL. In short, a *Drive* is a collection of functions (*open*, *read*, *close*, etc.) registered in LVGL to make file operations. You can add an interface to a standard file system (FAT32 on SD card) or you create your simple file system to read data from an SPI Flash memory. In every case, a *Drive* is just an abstraction to read and/or write data to a memory. See the [File system](#) section to learn more.

Images stored as files are not linked into the resulting executable, and must be read to RAM before being drawn. As a result, they are not as resource-friendly as variable images. However, they are easier to replace without needing to recompile the main program.

4.8.2 Color formats

Various built-in color formats are supported:

- **LV_IMG_CF_TRUE_COLOR** Simply stores the RGB colors (in whatever color depth LVGL is configured for).
- **LV_IMG_CF_TRUE_COLOR_ALPHA** Like **LV_IMG_CF_TRUE_COLOR** but it also adds an alpha (transparency) byte for every pixel.
- **LV_IMG_CF_TRUE_COLOR_CHROMA_KEYED** Like **LV_IMG_CF_TRUE_COLOR** but if a pixel has **LV_COLOR_TRANSP** (set in *lv_conf.h*) color the pixel will be transparent.
- **LV_IMG_CF_INDEXED_1/2/4/8BIT** Uses a palette with 2, 4, 16 or 256 colors and stores each pixel in 1, 2, 4 or 8 bits.
- **LV_IMG_CF_ALPHA_1/2/4/8BIT** Only stores the Alpha value on 1, 2, 4 or 8 bits. The pixels take the color of **style.image.color** and the set opacity. The source image has to be an alpha channel. This is ideal for bitmaps similar to fonts (where the whole image is one color but you'd like to be able to change it).

The bytes of the **LV_IMG_CF_TRUE_COLOR** images are stored in the following order.

For 32-bit color depth:

- Byte 0: Blue
- Byte 1: Green
- Byte 2: Red
- Byte 3: Alpha

For 16-bit color depth:

- Byte 0: Green 3 lower bit, Blue 5 bit
- Byte 1: Red 5 bit, Green 3 higher bit
- Byte 2: Alpha byte (only with **LV_IMG_CF_TRUE_COLOR_ALPHA**)

For 8-bit color depth:

- Byte 0: Red 3 bit, Green 3 bit, Blue 2 bit
- Byte 2: Alpha byte (only with **LV_IMG_CF_TRUE_COLOR_ALPHA**)

You can store images in a *Raw* format to indicate that, it's not a built-in color format and an external *Image decoder* needs to be used to decode the image.

- **LV_IMG_CF_RAW** Indicates a basic raw image (e.g. a PNG or JPG image).
- **LV_IMG_CF_RAW_ALPHA** Indicates that the image has alpha and an alpha byte is added for every pixel.
- **LV_IMG_CF_RAW_CHROME_KEYED** Indicates that the image is chrome keyed as described in **LV_IMG_CF_TRUE_COLOR_CHROMA_KEYED** above.

4.8.3 Add and use images

You can add images to LVGL in two ways:

- using the online converter
- manually create images

Online converter

The online Image converter is available here: <https://lvgl.io/tools/imageconverter>

Adding an image to LVGL via online converter is easy.

1. You need to select a *BMP*, *PNG* or *JPG* image first.
2. Give the image a name that will be used within LVGL.
3. Select the *Color format*.
4. Select the type of image you want. Choosing a binary will generate a **.bin** file that must be stored separately and read using the *file support*. Choosing a variable will generate a standard C file that can be linked into your project.
5. Hit the *Convert* button. Once the conversion is finished, your browser will automatically download the resulting file.

In the converter C arrays (variables), the bitmaps for all the color depths (1, 8, 16 or 32) are included in the C file, but only the color depth that matches **LV_COLOR_DEPTH** in *lv_conf.h* will actually be linked into the resulting executable.

In case of binary files, you need to specify the color format you want:

- RGB332 for 8-bit color depth
- RGB565 for 16-bit color depth
- RGB565 Swap for 16-bit color depth (two bytes are swapped)
- RGB888 for 32-bit color depth

Manually create an image

If you are generating an image at run-time, you can craft an image variable to display it using LVGL. For example:

```
uint8_t my_img_data[] = {0x00, 0x01, 0x02, ...};

static lv_img_dsc_t my_img_dsc = {
    .header.always_zero = 0,
    .header.w = 80,
    .header.h = 60,
    .data_size = 80 * 60 * LV_COLOR_DEPTH / 8,
    .header.cf = LV_IMG_CF_TRUE_COLOR,          /*Set the color format*/
    .data = my_img_data,
};
```

If the color format is `LV_IMG_CF_TRUE_COLOR_ALPHA` you can set `data_size` like `80 * 60 * LV_IMG_PX_SIZE_ALPHA_BYTE`.

Another (possibly simpler) option to create and display an image at run-time is to use the *Canvas* object.

Use images

The simplest way to use an image in LVGL is to display it with an *lv_img* object:

```
lv_obj_t * icon = lv_img_create(lv_scr_act(), NULL);

/*From variable*/
lv_img_set_src(icon, &my_icon_dsc);

/*From file*/
lv_img_set_src(icon, "S:my_icon.bin");
```

If the image was converted with the online converter, you should use `LV_IMG_DECLARE(my_icon_dsc)` to declare the image in the file where you want to use it.

4.8.4 Image decoder

As you can see in the *Color formats* section, LVGL supports several built-in image formats. In many cases, these will be all you need. LVGL doesn't directly support, however, generic image formats like PNG or JPG.

To handle non-built-in image formats, you need to use external libraries and attach them to LVGL via the *Image decoder* interface.

The image decoder consists of 4 callbacks:

- **info** get some basic info about the image (width, height and color format).
- **open** open the image: either store the decoded image or set it to `NULL` to indicate the image can be read line-by-line.
- **read** if *open* didn't fully open the image this function should give some decoded data (max 1 line) from a given position.
- **close** close the opened image, free the allocated resources.

You can add any number of image decoders. When an image needs to be drawn, the library will try all the registered image decoder until finding one which can open the image, i.e. knowing that format.

The `LV_IMG_CF_TRUE_COLOR...`, `LV_IMG_INDEXED...` and `LV_IMG_ALPHA...` formats (essentially, all non-RAW formats) are understood by the built-in decoder.

Custom image formats

The easiest way to create a custom image is to use the online image converter and set **Raw**, **Raw with alpha** or **Raw with chrome keyed** format. It will just take every byte of the binary file you uploaded and write it as the image "bitmap". You then need to attach an image decoder that will parse that bitmap and generate the real, renderable bitmap.

`header.cf` will be `LV_IMG_CF_RAW`, `LV_IMG_CF_RAW_ALPHA` or `LV_IMG_CF_RAW_CHROME_KEYED` accordingly. You should choose the correct format according to your needs: fully opaque image, use alpha channel or use chroma keying.

After decoding, the *raw* formats are considered *True color* by the library. In other words, the image decoder must decode the *Raw* images to *True color* according to the format described in `[#color-formats](Color formats)` section.

If you want to create a custom image, you should use `LV_IMG_CF_USER_ENCODED_0..7` color formats. However, the library can draw the images only in *True color* format (or *Raw* but finally it's supposed to be in *True color* format). So the `LV_IMG_CF_USER_ENCODED...` formats are not known by the library, therefore, they should be decoded to one of the known formats from `[#color-formats](Color formats)` section. It's possible to decode the image to a non-true color format first, for example, `LV_IMG_INDEXED_4BITS`, and then call the built-in decoder functions to convert it to *True color*.

With *User encoded* formats, the color format in the open function (`dsc->header.cf`) should be changed according to the new format.

Register an image decoder

Here's an example of getting LVGL to work with PNG images.

First, you need to create a new image decoder and set some functions to open/close the PNG files. It should look like this:

```
/*Create a new decoder and register functions */
lv_img_decoder_t * dec = lv_img_decoder_create();
lv_img_decoder_set_info_cb(dec, decoder_info);
lv_img_decoder_set_open_cb(dec, decoder_open);
lv_img_decoder_set_close_cb(dec, decoder_close);

/**
 * Get info about a PNG image
 * @param decoder pointer to the decoder where this function belongs
 * @param src can be file name or pointer to a C array
 * @param header store the info here
 * @return LV_RES_OK: no error; LV_RES_INV: can't get the info
 */
static lv_res_t decoder_info(lv_img_decoder_t * decoder, const void * src, lv_img_header_t * header)
{
    /*Check whether the type `src` is known by the decoder*/
    if(is_png(src) == false) return LV_RES_INV;

    /* Read the PNG header and find `width` and `height` */
    ...
}
```

(continues on next page)

(continued from previous page)

```

    header->cf = LV_IMG_CF_RAW_ALPHA;
    header->w = width;
    header->h = height;
}

/**
 * Open a PNG image and return the decoded image
 * @param decoder pointer to the decoder where this function belongs
 * @param dsc pointer to a descriptor which describes this decoding session
 * @return LV_RES_OK: no error; LV_RES_INV: can't get the info
 */
static lv_res_t decoder_open(lv_img_decoder_t * decoder, lv_img_decoder_dsc_t * dsc)
{
    /*Check whether the type `src` is known by the decoder*/
    if(is_png(src) == false) return LV_RES_INV;

    /*Decode and store the image. If `dsc->img_data` is `NULL`, the `read_line`
    ↪ function will be called to get the image data line-by-line*/
    dsc->img_data = my_png_decoder(src);

    /*Change the color format if required. For PNG usually 'Raw' is fine*/
    dsc->header.cf = LV_IMG_CF_...

    /*Call a built in decoder function if required. It's not required if `my_png_
    ↪ decoder` opened the image in true color format.*/
    lv_res_t res = lv_img_decoder_built_in_open(decoder, dsc);

    return res;
}

/**
 * Decode `len` pixels starting from the given `x`, `y` coordinates and store them in
    ↪ `buf`.
 * Required only if the "open" function can't open the whole decoded pixel array.
    ↪ (dsc->img_data == NULL)
 * @param decoder pointer to the decoder the function associated with
 * @param dsc pointer to decoder descriptor
 * @param x start x coordinate
 * @param y start y coordinate
 * @param len number of pixels to decode
 * @param buf a buffer to store the decoded pixels
 * @return LV_RES_OK: ok; LV_RES_INV: failed
 */
lv_res_t decoder_built_in_read_line(lv_img_decoder_t * decoder, lv_img_decoder_dsc_t
    ↪ dsc, lv_coord_t x,
                                     lv_coord_t y, lv_coord_t len, uint8_t
    ↪ * buf)
{
    /*With PNG it's usually not required*/

    /*Copy `len` pixels from `x` and `y` coordinates in True color format to `buf` */
}

```

(continues on next page)

(continued from previous page)

```

/**
 * Free the allocated resources
 * @param decoder pointer to the decoder where this function belongs
 * @param dsc pointer to a descriptor which describes this decoding session
 */
static void decoder_close(lv_img_decoder_t * decoder, lv_img_decoder_dsc_t * dsc)
{
    /*Free all allocated data*/

    /*Call the built-in close function if the built-in open/read_line was used*/
    lv_img_decoder_built_in_close(decoder, dsc);
}

```

So in summary:

- In **decoder_info**, you should collect some basic information about the image and store it in **header**.
- In **decoder_open**, you should try to open the image source pointed by **dsc->src**. Its type is already in **dsc->src_type == LV_IMG_SRC_FILE/VARIABLE**. If this format/type is not supported by the decoder, return **LV_RES_INV**. However, if you can open the image, a pointer to the decoded *True color* image should be set in **dsc->img_data**. If the format is known but, you don't want to decode while image (e.g. no memory for it) set **dsc->img_data = NULL** to call **read_line** to get the pixels.
- In **decoder_close** you should free all the allocated resources.
- **decoder_read** is optional. Decoding the whole image requires extra memory and some computational overhead. However, if can decode one line of the image without decoding the whole image, you can save memory and time. To indicate that, the *line read* function should be used, set **dsc->img_data = NULL** in the open function.

Manually use an image decoder

LVGL will use the registered image decoder automatically if you try and draw a raw image (i.e. using the **lv_img** object) but you can use them manually too. Create a **lv_img_decoder_dsc_t** variable to describe the decoding session and call **lv_img_decoder_open()**.

```

lv_res_t res;
lv_img_decoder_dsc_t dsc;
res = lv_img_decoder_open(&dsc, &my_img_dsc, LV_COLOR_WHITE);

if(res == LV_RES_OK) {
    /*Do something with `dsc->img_data`*/
    lv_img_decoder_close(&dsc);
}

```

4.8.5 Image caching

Sometimes it takes a lot of time to open an image. Continuously decoding a PNG image or loading images from a slow external memory would be inefficient and detrimental to the user experience.

Therefore, LVGL caches a given number of images. Caching means some images will be left open, hence LVGL can quickly access them from `dsc->img_data` instead of needing to decode them again.

Of course, caching images is resource-intensive as it uses more RAM (to store the decoded image). LVGL tries to optimize the process as much as possible (see below), but you will still need to evaluate if this would be beneficial for your platform or not. If you have a deeply embedded target which decodes small images from a relatively fast storage medium, image caching may not be worth it.

Cache size

The number of cache entries can be defined in `LV_IMG_CACHE_DEF_SIZE` in `lv_conf.h`. The default value is 1 so only the most recently used image will be left open.

The size of the cache can be changed at run-time with `lv_img_cache_set_size(entry_num)`.

Value of images

When you use more images than cache entries, LVGL can't cache all of the images. Instead, the library will close one of the cached images (to free space).

To decide which image to close, LVGL uses a measurement it previously made of how long it took to open the image. Cache entries that hold slower-to-open images are considered more valuable and are kept in the cache as long as possible.

If you want or need to override LVGL's measurement, you can manually set the *time to open* value in the decoder open function in `dsc->time_to_open = time_ms` to give a higher or lower value. (Leave it unchanged to let LVGL set it.)

Every cache entry has a *"life"* value. Every time an image opening happens through the cache, the *life* of all entries are decreased to make them older. When a cached image is used, its *life* is increased by the *time to open* value to make it more alive.

If there is no more space in the cache, always the entry with the smallest life will be closed.

Memory usage

Note that, the cached image might continuously consume memory. For example, if 3 PNG images are cached, they will consume memory while they are opened.

Therefore, it's the user's responsibility to be sure there is enough RAM to cache, even the largest images at the same time.

Clean the cache

Let's say you have loaded a PNG image into a `lv_img_dsc_t my_png` variable and use it in an `lv_img` object. If the image is already cached and you then change the underlying PNG file, you need to notify LVGL to cache the image again. Otherwise, there is no easy way of detecting that the underlying file changed and LVGL will still draw the old image.

To do this, use `lv_img_cache_invalidate_src(&my_png)`. If `NULL` is passed as a parameter, the whole cache will be cleaned.

4.9 File system

LVGL has a 'File system' abstraction module that enables you to attach any type of file systems. The file system is identified by a drive letter. For example, if the SD card is associated with the letter 'S', a file can be reached like "S:path/to/file.txt".

4.9.1 Add a driver

To add a driver, `lv_fs_drv_t` needs to be initialized like this:

```
lv_fs_drv_t drv;
lv_fs_drv_init(&drv);                                /*Basic initialization*/

drv.letter = 'S';                                     /*An uppercase letter to identify the drive_
↳*/
drv.file_size = sizeof(my_file_object);              /*Size required to store a file object*/
drv.rddir_size = sizeof(my_dir_object);              /*Size required to store a directory object_
↳(used by dir_open/close/read)*/
drv.ready_cb = my_ready_cb;                          /*Callback to tell if the drive is ready to_
↳use */
drv.open_cb = my_open_cb;                            /*Callback to open a file */
drv.close_cb = my_close_cb;                         /*Callback to close a file */
drv.read_cb = my_read_cb;                          /*Callback to read a file */
drv.write_cb = my_write_cb;                        /*Callback to write a file */
drv.seek_cb = my_seek_cb;                          /*Callback to seek in a file (Move cursor)_
↳*/
drv.tell_cb = my_tell_cb;                          /*Callback to tell the cursor position */
drv.trunc_cb = my_trunc_cb;                        /*Callback to delete a file */
drv.size_cb = my_size_cb;                          /*Callback to tell a file's size */
drv.rename_cb = my_rename_cb;                      /*Callback to rename a file */

drv.dir_open_cb = my_dir_open_cb;                  /*Callback to open directory to read its_
↳content */
drv.dir_read_cb = my_dir_read_cb;                  /*Callback to read a directory's content */
drv.dir_close_cb = my_dir_close_cb;                /*Callback to close a directory */

drv.free_space_cb = my_free_space_cb;              /*Callback to tell free space on the drive_
↳*/

drv.user_data = my_user_data;                      /*Any custom data if required*/

lv_fs_drv_register(&drv);                          /*Finally register the drive*/
```

Any of the callbacks can be `NULL` to indicate that that operation is not supported.

As an example of how the callbacks are used, if you use `lv_fs_open(&file, "S:/folder/file.txt", LV_FS_MODE_WR)`, LVGL:

1. Verifies that a registered drive exists with the letter 'S'.
2. Checks if it's `open_cb` is implemented (not `NULL`).
3. Calls the set `open_cb` with "folder/file.txt" path.

4.9.2 Usage example

The example below shows how to read from a file:

```
lv_fs_file_t f;
lv_fs_res_t res;
res = lv_fs_open(&f, "S:/folder/file.txt", LV_FS_MODE_RD);
if(res != LV_FS_RES_OK) my_error_handling();

uint32_t read_num;
uint8_t buf[8];
res = lv_fs_read(&f, buf, 8, &read_num);
if(res != LV_FS_RES_OK || read_num != 8) my_error_handling();

lv_fs_close(&f);
```

The mode in `lv_fs_open` can be `LV_FS_MODE_WR` to open for write or `LV_FS_MODE_RD` | `LV_FS_MODE_WR` for both

This example shows how to read a directory's content. It's up to the driver how to mark the directories, but it can be a good practice to insert a '/' in front of the directory name.

```
lv_fs_dir_t dir;
lv_fs_res_t res;
res = lv_fs_dir_open(&dir, "S:/folder");
if(res != LV_FS_RES_OK) my_error_handling();

char fn[256];
while(1) {
    res = lv_fs_dir_read(&dir, fn);
    if(res != LV_FS_RES_OK) {
        my_error_handling();
        break;
    }

    /*fn is empty, if not more files to read*/
    if(strlen(fn) == 0) {
        break;
    }

    printf("%s\n", fn);
}

lv_fs_dir_close(&dir);
```

4.9.3 Use drivers for images

Image objects can be opened from files too (besides variables stored in the flash).

To initialize the image, the following callbacks are required:

- open
- close
- read
- seek
- tell

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 LVGL. 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.

```
/* INITIALIZE AN ANIMATION
 *-----*/

lv_anim_t a;
lv_anim_init(&a);

/* MANDATORY SETTINGS
 *-----*/

/*Set the "animator" function*/
lv_anim_set_exec_cb(&a, (lv_anim_exec_xcb_t) lv_obj_set_x);

/*Set the "animator" function*/
lv_anim_set_var(&a, obj);

/*Length of the animation [ms]*/
lv_anim_set_time(&a, duration);

/*Set start and end values. E.g. 0, 150*/
lv_anim_set_values(&a, start, end);
```

(continues on next page)

(continued from previous page)

```

/* OPTIONAL SETTINGS
 *-----*/

/*Time to wait before starting the animation [ms]*/
lv_anim_set_delay(&a, delay);

/*Set path (curve). Default is linear*/
lv_anim_set_path(&a, &path);

/*Set a callback to call when animation is ready.*/
lv_anim_set_ready_cb(&a, ready_cb);

/*Set a callback to call when animation is started (after delay).*/
lv_anim_set_start_cb(&a, start_cb);

/*Play the animation backward too with this duration. Default is 0 (disabled) [ms]*/
lv_anim_set_playback_time(&a, wait_time);

/*Delay before playback. Default is 0 (disabled) [ms]*/
lv_anim_set_playback_delay(&a, wait_time);

/*Number of repetitions. Default is 1. LV_ANIM_REPEAT_INFINIT for infinite.
↳repetition*/
lv_anim_set_repeat_count(&a, wait_time);

/*Delay before repeat. Default is 0 (disabled) [ms]*/
lv_anim_set_repeat_delay(&a, wait_time);

/*true (default): apply the start vale immediately, false: apply start vale after.
↳delay when then anim. really starts. */
lv_anim_set_early_apply(&a, true/false);

/* START THE ANIMATION
 *-----*/
lv_anim_start(&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_start()` 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 mainly 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 functions:

- `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)

A path can be initialized like this:

```
lv_anim_path_t path;
lv_anim_path_init(&path);
lv_anim_path_set_cb(&path, lv_anim_path_overshoot);
lv_anim_path_set_user_data(&path, &foo); /*Optional for custom functions*/

/*Set the path in an animation*/
lv_anim_set_path(&a, &path);
```

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 Tasks

LVGL has a built-in task system. You can register a function to have it be called periodically. The tasks are handled and called in `lv_task_handler()`, which needs to be called periodically every few milliseconds. See *Porting* for more information.

The tasks are non-preemptive, which means a task cannot interrupt another task. Therefore, you can call any LVGL related function in a task.

4.11.1 Create a task

To create a new task, use `lv_task_create(task_cb, period_ms, LV_TASK_PRIO_OFF/LOWEST/LOW/MID/HIGH/HIGHEST, user_data)`. It will create an `lv_task_t *` variable, which can be used later to modify the parameters of the task. `lv_task_create_basic()` can also be used. It allows you to create a new task without specifying any parameters.

A task callback should have `void (*lv_task_cb_t)(lv_task_t *)`; prototype.

For example:

```
void my_task(lv_task_t * task)
{
    /*Use the user_data*/
    uint32_t * user_data = task->user_data;
    printf("my_task called with user data: %d\n", *user_data);
}
```

(continues on next page)

(continued from previous page)

```

/*Do something with LVGL*/
if(something_happened) {
    something_happened = false;
    lv_btn_create(lv_scr_act(), NULL);
}
}

...

static uint32_t user_data = 10;
lv_task_t * task = lv_task_create(my_task, 500, LV_TASK_PRIO_MID, &user_data);

```

4.11.2 Ready and Reset

`lv_task_ready(task)` makes the task run on the next call of `lv_task_handler()`.

`lv_task_reset(task)` resets the period of a task. It will be called again after the defined period of milliseconds has elapsed.

4.11.3 Set parameters

You can modify some parameters of the tasks later:

- `lv_task_set_cb(task, new_cb)`
- `lv_task_set_period(task, new_period)`
- `lv_task_set_prio(task, new_priority)`

4.11.4 One-shot tasks

You can make a task to run only once by calling `lv_task_once(task)`. The task will automatically be deleted after being called for the first time.

4.11.5 Measure idle time

You can get the idle percentage time `lv_task_handler` with `lv_task_get_idle()`. Note that, it doesn't measure the idle time of the overall system, only `lv_task_handler`. It can be misleading if you use an operating system and call `lv_task_handler` in a task, as it won't actually measure the time the OS spends in an idle thread.

4.11.6 Asynchronous calls

In some cases, you can't do an action immediately. For example, you can't delete an object right now because something else is still using it or you don't want to block the execution now. For these cases, you can use the `lv_async_call(my_function, data_p)` to make `my_function` be called on the next call of `lv_task_handler`. `data_p` will be passed to function when it's called. Note that, only the pointer of the data is saved so you need to ensure that the variable will be "alive" while the function is called. You can use *static*, global or dynamically allocated data.

For example:

```

void my_screen_clean_up(void * scr)
{
    /*Free some resources related to `scr`*/

    /*Finally delete the screen*/
    lv_obj_del(scr);
}

...

/*Do somethings with the object on the current screen*/

/*Delete screen on next call of `lv_task_handler`. So not now.*/
lv_async_call(my_screen_clean_up, lv_scr_act());

/*The screen is still valid so you can do other things with it*/

```

If you just want to delete an object, and don't need to clean anything up in `my_screen_clean_up`, you could just use `lv_obj_del_async`, which will delete the object on the next call to `lv_task_handler`.

4.12 Drawing

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

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

The basic concept is to not draw directly to the 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. For example, when drawing a *background + button + text*, each "stage" would be visible for a short time.
2. **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). Hence, it's suitable for pixels that are redrawn multiple times (e.g. background + button + text).

4.12.1 Buffering types

As you already might learn in the *Porting* section, there are 3 types of buffers:

1. **One buffer** - LVGL 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, LVGL 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. If the buffer is smaller than the area to refresh, LVGL will draw the display's content in chunks similar to the *One buffer*.
3. **Two screen-sized buffers** - In contrast to *Two non-screen-sized buffers*, LVGL 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 LVGL. Therefore, this method works best when the MCU has an LCD/TFT interface and the frame buffer is just a location in the RAM.

4.12.2 Mechanism of screen refreshing

1. Something happens on the GUI which requires redrawing. For example, a button has been pressed, a chart has been changed or an animation happened, etc.
2. LVGL saves the changed object's old and new area into a buffer, called an *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*):
 - LVGL checks the invalid areas and joins the adjacent or intersecting areas.
 - Takes the first joined area, if it's smaller than the *display buffer*, then simply draw 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** - LVGL 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** - LVGL 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.

4.12.3 Masking

Masking is the basic concept of LVGL's drawing engine. To use LVGL it's not required to know about the mechanisms described here, but you might find interesting to know how the drawing works under hood.

To learn masking let's learn the steps of drawing first:

1. Create a draw descriptor from an object's styles (e.g. `lv_draw_rect_dsc_t`). It tells the parameters of drawing, for example the colors, widths, opacity, fonts, radius, etc.
2. Call the draw function with the initialized descriptor and some other parameters. It renders the primitive shape to the current draw buffer.
3. If the shape is very simple and doesn't require masks go to #5. Else create the required masks (e.g. a rounded rectangle mask)

4. Apply all the created mask(s) for one or a few lines. It create 0..255 values into a *mask buffer* with the "shape" of the created masks. E.g. in case of a "line mask" according to the parameters of the mask, keep one side of the buffer as it is (255 by default) and set the rest to 0 to indicate that the latter side should be removed.
5. Blend the image or rectangle to the screen. During blending masks (make some pixels transparent or opaque), blending modes (additive, subtractive, etc), opacity are handled.
6. Repeat from #4.

Masks are used the create almost every basic primitives:

- **letters** create a mask from the letter and draw a "letter-colored" rectangle using the mask.
- **line** created from 4 "line masks", to mask out the left, right, top and bottom part of the line to get perfectly perpendicular line ending
- **rounded rectangle** a mask is created real-time for each line of a rounded rectangle and a normal filled rectangle is drawn according to the mask.
- **clip corner** to clip to overflowing content on the rounded corners also a rounded rectangle mask is applied.
- **rectangle border** same as a rounded rectangle, but inner part is masked out too
- **arc drawing** a circle border is drawn, but an arc mask is applied.
- **ARGB images** the alpha channel is separated into a mask and the image is drawn as a normal RGB image.

As mentioned in #3 above in some cases no mask is required:

- a mono colored, not rounded rectangles
- RGB images

LVGL has the following built-in mask types which can be calculated and applied real-time:

- **LV_DRAW_MASK_TYPE_LINE** Removes a side of a line (top, bottom, left or right). `lv_draw_line` uses 4 of it. Essentially, every (skew) line is bounded with 4 line masks by forming a rectangle.
- **LV_DRAW_MASK_TYPE_RADIUS** Removes the inner or outer parts of a rectangle which can have radius too. It's also used to create circles by setting the radius to large value (`LV_RADIUS_CIRCLE`)
- **LV_DRAW_MASK_TYPE_ANGLE** Removes a circle sector. It is used by `lv_draw_arc` to remove the "empty" sector.
- **LV_DRAW_MASK_TYPE_FADE** Create a vertical fade (change opacity)
- **LV_DRAW_MASK_TYPE_MAP** The mask is stored in an array and the necessary parts are applied

Masks are create and removed automatically during drawing but the *lv_objmask* allows the user to add masks. Here is an example:

C

Several object masks

code

```

#include "../../lv_examples.h"
#if LV_USE_OBJMASK

void lv_ex_objmask_1(void)
{
    /*Set a very visible color for the screen to clearly see what happens*/
    lv_obj_set_style_local_bg_color(lv_scr_act(), LV_OBJ_PART_MAIN, LV_STATE_DEFAULT,
↪ lv_color_hex3(0xf33));

    lv_obj_t * om = lv_objmask_create(lv_scr_act(), NULL);
    lv_obj_set_size(om, 200, 200);
    lv_obj_align(om, NULL, LV_ALIGN_CENTER, 0, 0);
    lv_obj_t * label = lv_label_create(om, NULL);
    lv_label_set_long_mode(label, LV_LABEL_LONG_BREAK);
    lv_label_set_align(label, LV_LABEL_ALIGN_CENTER);
    lv_obj_set_width(label, 180);
    lv_label_set_text(label, "This label will be masked out. See how it works.");
    lv_obj_align(label, NULL, LV_ALIGN_IN_TOP_MID, 0, 20);

    lv_obj_t * cont = lv_cont_create(om, NULL);
    lv_obj_set_size(cont, 180, 100);
    lv_obj_set_drag(cont, true);
    lv_obj_align(cont, NULL, LV_ALIGN_IN_BOTTOM_MID, 0, -10);

    lv_obj_t * btn = lv_btn_create(cont, NULL);
    lv_obj_align(btn, NULL, LV_ALIGN_CENTER, 0, 0);
    lv_obj_set_style_local_value_str(btn, LV_BTN_PART_MAIN, LV_STATE_DEFAULT, "Button
↪");
    uint32_t t;

    lv_refr_now(NULL);
    t = lv_tick_get();
    while(lv_tick_elaps(t) < 1000);

    lv_area_t a;
    lv_draw_mask_radius_param_t r1;

    a.x1 = 10;
    a.y1 = 10;
    a.x2 = 190;
    a.y2 = 190;
    lv_draw_mask_radius_init(&r1, &a, LV_RADIUS_CIRCLE, false);
    lv_objmask_add_mask(om, &r1);

    lv_refr_now(NULL);
    t = lv_tick_get();
    while(lv_tick_elaps(t) < 1000);

    a.x1 = 100;
    a.y1 = 100;

```

(continues on next page)

(continued from previous page)

```

a.x2 = 150;
a.y2 = 150;
lv_draw_mask_radius_init(&r1, &a, LV_RADIUS_CIRCLE, true);
lv_objmask_add_mask(om, &r1);

lv_refr_now(NULL);
t = lv_tick_get();
while(lv_tick_elaps(t) < 1000);

lv_draw_mask_line_param_t l1;
lv_draw_mask_line_points_init(&l1, 0, 0, 100, 200, LV_DRAW_MASK_LINE_SIDE_TOP);
lv_objmask_add_mask(om, &l1);

lv_refr_now(NULL);
t = lv_tick_get();
while(lv_tick_elaps(t) < 1000);

lv_draw_mask_fade_param_t f1;
a.x1 = 100;
a.y1 = 0;
a.x2 = 200;
a.y2 = 200;
lv_draw_mask_fade_init(&f1, &a, LV_OPA_TRANSP, 0, LV_OPA_COVER, 150);
lv_objmask_add_mask(om, &f1);
}

#endif

```

Text mask

code

```

#include "../../lv_examples.h"
#if LV_USE_OBJMASK

#define MASK_WIDTH 100
#define MASK_HEIGHT 50

void lv_ex_objmask_2(void)
{
    /* Create the mask of a text by drawing it to a canvas*/
    static lv_opa_t mask_map[MASK_WIDTH * MASK_HEIGHT];

    /*Create a "8 bit alpha" canvas and clear it*/
    lv_obj_t * canvas = lv_canvas_create(lv_scr_act(), NULL);
    lv_canvas_set_buffer(canvas, mask_map, MASK_WIDTH, MASK_HEIGHT, LV_IMG_CF_ALPHA_
↪ 8BIT);
    lv_canvas_fill_bg(canvas, LV_COLOR_BLACK, LV_OPA_TRANSP);

    /*Draw a label to the canvas. The result "image" will be used as mask*/
    lv_draw_label_dsc_t label_dsc;
    lv_draw_label_dsc_init(&label_dsc);
    label_dsc.color = LV_COLOR_WHITE;
    lv_canvas_draw_text(canvas, 5, 5, MASK_WIDTH, &label_dsc, "Text with gradient",
↪ LV_LABEL_ALIGN_CENTER);

```

(continues on next page)

(continued from previous page)

```

/*The mask is reads the canvas is not required anymore*/
lv_obj_del(canvas);

/*Create an object mask which will use the created mask*/
lv_obj_t * om = lv_objmask_create(lv_scr_act(), NULL);
lv_obj_set_size(om, MASK_WIDTH, MASK_HEIGHT);
lv_obj_align(om, NULL, LV_ALIGN_CENTER, 0, 0);

/*Add the created mask map to the object mask*/
lv_draw_mask_map_param_t m;
lv_area_t a;
a.x1 = 0;
a.y1 = 0;
a.x2 = MASK_WIDTH - 1;
a.y2 = MASK_HEIGHT - 1;
lv_draw_mask_map_init(&m, &a, mask_map);
lv_objmask_add_mask(om, &m);

/*Create a style with gradient*/
static lv_style_t style_bg;
lv_style_init(&style_bg);
lv_style_set_bg_opa(&style_bg, LV_STATE_DEFAULT, LV_OPA_COVER);
lv_style_set_bg_color(&style_bg, LV_STATE_DEFAULT, LV_COLOR_RED);
lv_style_set_bg_grad_color(&style_bg, LV_STATE_DEFAULT, LV_COLOR_BLUE);
lv_style_set_bg_grad_dir(&style_bg, LV_STATE_DEFAULT, LV_GRAD_DIR_HOR);

/* Create and object with the gradient style on the object mask.
 * The text will be masked from the gradient*/
lv_obj_t * bg = lv_obj_create(om, NULL);
lv_obj_reset_style_list(bg, LV_OBJ_PART_MAIN);
lv_obj_add_style(bg, LV_OBJ_PART_MAIN, &style_bg);
lv_obj_set_size(bg, MASK_WIDTH, MASK_HEIGHT);
}

#endif

```

MicroPython

No examples yet.

WIDGETS

5.1 Base object (lv_obj)

5.1.1 Overview

The 'Base Object' implements the basic properties of widgets on a screen, such as:

- coordinates
- parent object
- children
- main style
- attributes like *Click enable*, *Drag enable*, etc.

In object-oriented thinking, it is the base class from which all other objects in LVGL are inherited. This, among another things, helps reduce code duplication.

The functions and functionalities of Base object can be used with other widgets too. For example `lv_obj_set_width(slider, 100)`

The Base object can be directly used as a simple widgets. It nothing else then a rectangle.

Coordinates

Size

The object size can be modified on individual axes with `lv_obj_set_width(obj, new_width)` and `lv_obj_set_height(obj, new_height)`, or both axes can be modified at the same time with `lv_obj_set_size(obj, new_width, new_height)`.

Styles can add [Margin](#) to the objects. Margin tells that "I want this space around me". To set width or height reduced by the margin `lv_obj_set_width_margin(obj, new_width)` or `lv_obj_set_height_margin(obj, new_height)`. In more exact way: `new_width = left_margin + object_width + right_margin`.

To get the width or height which includes the margins use `lv_obj_get_width/height_margin(obj)`.

Styles can add [Padding](#) to the object as well. Padding means "I don't want my children too close to my sides, so keep this space". To set width or height reduced by the padding `lv_obj_set_width_fit(obj, new_width)` or `lv_obj_set_height_fit(obj, new_height)`. In a more exact way: `new_width = left_pad + object_width + right_pad` To get the width or height which is REDUCED by padding use `lv_obj_get_width/height_fit(obj)`. It can be considered the "useful size of the object".

Margin and padding gets important when [Layout](#) or [Auto-fit](#) is used by other widgets.

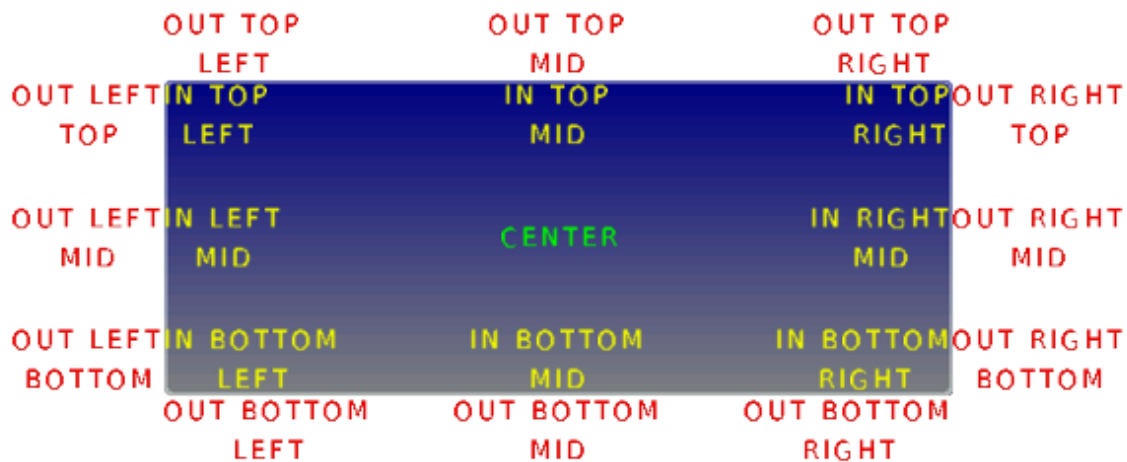
Position

You can set the x and y coordinates relative to the parent with `lv_obj_set_x(obj, new_x)` and `lv_obj_set_y(obj, new_y)`, or both at the same time with `lv_obj_set_pos(obj, new_x, new_y)`.

Alignment

You can align the object to another with `lv_obj_align(obj, obj_ref, LV_ALIGN_..., x_ofs, y_ofs)`.

- `obj` is the object to align.
- `obj_ref` is a reference object. `obj` will be aligned to it. If `obj_ref = NULL`, then the parent of `obj` will be used.
- The third argument is the *type* of alignment. These are the possible options:



The alignment types build like `LV_ALIGN_OUT_TOP_MID`.

- The last two arguments allow you to shift the object by a specified number of pixels after aligning it.

For example, to align a text below an image: `lv_obj_align(text, image, LV_ALIGN_OUT_BOTTOM_MID, 0, 10)`. Or to align a text in the middle of its parent: `lv_obj_align(text, NULL, LV_ALIGN_CENTER, 0, 0)`.

`lv_obj_align_origo` works similarly to `lv_obj_align` but it aligns the center of the object.

For example, `lv_obj_align_origo(btn, image, LV_ALIGN_OUT_BOTTOM_MID, 0, 0)` will align the center of the button the bottom of the image.

The parameters of the alignment will be saved in the object if `LV_USE_OBJ_REALIGN` is enabled in `lv_conf.h`. You can then realign the objects simply by calling `lv_obj_realign(obj)`. It's equivalent to calling `lv_obj_align` again with the same parameters.

If the alignment happened with `lv_obj_align_origo`, then it will be used when the object is realigned.

The `lv_obj_align_x/y` and `lv_obj_align_origo_x/y` function can be used to align only on one axis.

If `lv_obj_set_auto_realign(obj, true)` is used the object will be realigned automatically, if its size changes in `lv_obj_set_width/height/size()` functions. It's very useful when size animations are applied to the object and the original position needs to be kept.

Note that the coordinates of screens can't be changed. Attempting to use these functions on screens will result in undefined behavior.

Parents and children

You can set a new parent for an object with `lv_obj_set_parent(obj, new_parent)`. To get the current parent, use `lv_obj_get_parent(obj)`.

To get the children of an object, use `lv_obj_get_child(obj, child_prev)` (from last to first) or `lv_obj_get_child_back(obj, child_prev)` (from first to last). To get the first child, pass `NULL` as the second parameter and use the return value to iterate through the children. The function will return `NULL` if there are no more children. For example:

```
lv_obj_t * child = lv_obj_get_child(parent, NULL);
while(child) {
    /*Do something with "child" */
    child = lv_obj_get_child(parent, child);
}
```

`lv_obj_count_children(obj)` tells the number of children on an object. `lv_obj_count_children_recursive(obj)` also tells the number of children but counts children of children recursively.

Screens

When you have created a screen like `lv_obj_t * screen = lv_obj_create(NULL, NULL)`, you can load it with `lv_scr_load(screen)`. The `lv_scr_act()` function gives you a pointer to the current screen.

If you have more display then it's important to know that these functions operate on the lastly created or the explicitly selected (with `lv_disp_set_default`) display.

To get an object's screen use the `lv_obj_get_screen(obj)` function.

Layers

There are two automatically generated layers:

- top layer
- system layer

They are independent of the screens and they will be shown on every screen. The *top layer* is above every object on the screen and the *system layer* is above the *top layer* too. You can add any pop-up windows to the *top layer* freely. But, the *system layer* is restricted to system-level things (e.g. mouse cursor will be placed here in `lv_indev_set_cursor()`).

The `lv_layer_top()` and `lv_layer_sys()` functions gives a pointer to the top or system layer.

You can bring an object to the foreground or send it to the background with `lv_obj_move_foreground(obj)` and `lv_obj_move_background(obj)`.

Read the [Layer overview](#) section to learn more about layers.

Events

To set an event callback for an object, use `lv_obj_set_event_cb(obj, event_cb)`,

To manually send an event to an object, use `lv_event_send(obj, LV_EVENT_..., data)`

Read the [Event overview](#) to learn more about the events.

5.1.2 Parts

The widgets can have multiple parts. For example a [Button](#) has only a main part but a [Slider](#) is built from a background, an indicator and a knob.

The name of the parts is constructed like `LV_ + <TYPE> _PART_ <NAME>`. For example `LV_BTN_PART_MAIN` or `LV_SLIDER_PART_KNOB`. The parts are usually used when styles are added to the objects. Using parts different styles can be assigned to the different parts of the objects.

To learn more about the parts read the related section of the [Style overview](#).

States

The object can be in a combinations of the following states:

- **LV_STATE_DEFAULT** Normal, released
- **LV_STATE_CHECKED** Toggled or checked
- **LV_STATE_FOCUSED** Focused via keypad or encoder or clicked via touchpad/mouse
- **LV_STATE_EDITED** Edit by an encoder
- **LV_STATE_HOVERED** Hovered by mouse (not supported now)
- **LV_STATE_PRESSED** Pressed
- **LV_STATE_DISABLED** Disabled or inactive

The states are usually automatically changed by the library as the user presses, releases, focuses etc an object. However, the states can be changed manually too. To completely overwrite the current state use `lv_obj_set_state(obj, part, LV_STATE_...)`. To set or clear given state (but leave to other states untouched) use `lv_obj_add/clear_state(obj, part, LV_STATE_...)` In both cases ORed state values can be used as well. E.g. `lv_obj_set_state(obj, part, LV_STATE_PRESSED | LV_STATE_PRESSED_CHECKED)`.

To learn more about the states read the related section of the [Style overview](#).

Style

Be sure to read the [Style overview](#) first.

To add a style to an object use `lv_obj_add_style(obj, part, &new_style)` function. The Base object use all the rectangle-like style properties.

To remove all styles from an object use `lv_obj_reset_style_list(obj, part)`

If you modify a style, which is already used by objects, in order to refresh the affected objects you can use either `lv_obj_refresh_style(obj)` on each object using it or to notify all objects with a given style use `lv_obj_report_style_mod(&style)`. If the parameter of `lv_obj_report_style_mod` is `NULL`, all objects will be notified.

Attributes

There are some attributes which can be enabled/disabled by `lv_obj_set...(obj, true/false)`:

- **hidden** - Hide the object. It will not be drawn and will be considered by input devices as if it doesn't exist., Its children will be hidden too.
- **click** - Allows you to click the object via input devices. If disabled, then click events are passed to the object behind this one. (E.g. *Labels* are not clickable by default)
- **top** - If enabled then when this object or any of its children is clicked then this object comes to the foreground.
- **drag** - Enable dragging (moving by an input device)
- **drag_dir** - Enable dragging only in specific directions. Can be `LV_DRAG_DIR_HOR/VER/ALL`.
- **drag_throw** - Enable "throwing" with dragging as if the object would have momentum
- **drag_parent** - If enabled then the object's parent will be moved during dragging. It will look like as if the parent is dragged. Checked recursively, so can propagate to grandparents too.
- **parent_event** - Propagate the events to the parents too. Checked recursively, so can propagate to grandparents too.
- **opa_scale_enable** - Enable opacity scaling. See the `[#opa-scale](Opa scale)` section.

Protect

There are some specific actions which happen automatically in the library. To prevent one or more that kind of actions, you can protect the object against them. The following protections exists:

- **LV_PROTECT_NONE** No protection
- **LV_PROTECT_POS** Prevent automatic positioning (e.g. Layout in *Containers*)
- **LV_PROTECT_FOLLOW** Prevent the object be followed (make a "line break") in automatic ordering (e.g. Layout in *Containers*)
- **LV_PROTECT_PARENT** Prevent automatic parent change. (e.g. *Page* moves the children created on the background to the scrollable)
- **LV_PROTECT_PRESS_LOST** Prevent losing press when the press is slid out of the objects. (E.g. a *Button* can be released out of it if it was being pressed)
- **LV_PROTECT_CLICK_FOCUS** Prevent automatically focusing the object if it's in a *Group* and click focus is enabled.
- **LV_PROTECT_CHILD_CHG** Disable the child change signal. Used internally by the library

The `lv_obj_add/clear_protect(obj, LV_PROTECT...)` sets/clears the protection. You can use 'OR'ed values of protection types too.

Groups

Once, an object is added to *group* with `lv_group_add_obj(group, obj)` the object's current group can be get with `lv_obj_get_group(obj)`.

`lv_obj_is_focused(obj)` tells if the object is currently focused on its group or not. If the object is not added to a group, `false` will be returned.

Read the *Input devices overview* to learn more about the *Groups*.

Extended click area

By default, the objects can be clicked only on their coordinates, however, this area can be extended with `lv_obj_set_ext_click_area(obj, left, right, top, bottom)`. `left/right/top/bottom` describes how far the clickable area should extend past the default in each direction.

This feature needs to be enabled in *lv_conf.h* with `LV_USE_EXT_CLICK_AREA`. The possible values are:

- `LV_EXT_CLICK_AREA_FULL` store all 4 coordinates as `lv_coord_t`
- `LV_EXT_CLICK_AREA_TINY` store only horizontal and vertical coordinates (use the greater value of left/right and top/bottom) as `uint8_t`
- `LV_EXT_CLICK_AREA_OFF` Disable this feature

5.1.3 Events

Only the *Generic events* are sent by the object type.

Learn more about *Events*.

5.1.4 Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

5.1.5 Example

C

Base objects with custom styles

code

```
#include "../../lv_examples.h"

void lv_ex_obj_1(void)
{
    lv_obj_t * obj1;
    obj1 = lv_obj_create(lv_scr_act(), NULL);
    lv_obj_set_size(obj1, 100, 50);
    lv_obj_align(obj1, NULL, LV_ALIGN_CENTER, -60, -30);
}
```

(continues on next page)

(continued from previous page)

```

/*Copy the previous object and enable drag*/
lv_obj_t * obj2;
obj2 = lv_obj_create(lv_scr_act(), obj1);
lv_obj_align(obj2, NULL, LV_ALIGN_CENTER, 0, 0);
lv_obj_set_drag(obj2, true);

static lv_style_t style_shadow;
lv_style_init(&style_shadow);
lv_style_set_shadow_width(&style_shadow, LV_STATE_DEFAULT, 10);
lv_style_set_shadow_spread(&style_shadow, LV_STATE_DEFAULT, 5);
lv_style_set_shadow_color(&style_shadow, LV_STATE_DEFAULT, LV_COLOR_BLUE);

/*Copy the previous object (drag is already enabled)*/
lv_obj_t * obj3;
obj3 = lv_obj_create(lv_scr_act(), obj2);
lv_obj_add_style(obj3, LV_OBJ_PART_MAIN, &style_shadow);
lv_obj_align(obj3, NULL, LV_ALIGN_CENTER, 60, 30);
}

```

MicroPython

No examples yet.

5.2 Arc (lv_arc)

5.2.1 Overview

The Arc are consists of a background and a foreground arc. Both can have start and end angles and thickness.

5.2.2 Parts and Styles

The Arc's main part is called **LV_ARC_PART_MAIN**. It draws a background using the typical background style properties and an arc using the *line* style properties. The arc's size and position will respect the *padding* style properties.

LV_ARC_PART_INDIC is virtual part and it draws an other arc using the *line* style properties. It's padding values are interpreted relative to the background arc. The radius of the indicator arc will be modified according to the greatest padding value.

LV_ARC_PART_KNOB is virtual part and it draws on the end of the arc indicator. It uses all background properties and padding values. With zero padding the knob size is the same as the indicator's width. Larger padding makes it larger, smaller padding makes it smaller.

5.2.3 Usage

Angles

To set the angles of the background, use the `lv_arc_set_bg_angles(arc, start_angle, end_angle)` function or `lv_arc_set_bg_start/end_angle(arc, start_angle)`. Zero degree is at the middle right (3 o'clock) of the object and the degrees are increasing in a clockwise direction. The angles should be in [0;360] range.

Similarly, `lv_arc_set_angles(arc, start_angle, end_angle)` function or `lv_arc_set_start/end_angle(arc, start_angle)` sets the angles of the indicator arc.

Rotation

An offset to the 0 degree position can added with `lv_arc_set_rotation(arc, deg)`.

Range and values

Besides setting angles manually the arc can have a range and a value. To set the range use `lv_arc_set_range(arc, min, max)` and to set a value use `lv_arc_set_value(arc, value)`. Using range and value the angle of the indicator will be mapped between background angles.

Type

The arc can have the different "types". They are used only the value is set by `lv_arc_set_value`. The following types exist:

- `LV_ARC_TYPE_NORMAL` indicator arc drawn clockwise (min to current)
- `LV_ARC_TYPE_REVERSE` indicator arc drawn counter clockwise (max to current)
- `LV_ARC_TYPE_SYMMETRIC` indicator arc drawn from the middle point to the current value.

5.2.4 Events

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

- `LV_EVENT_VALUE_CHANGED` sent when the arc is pressed/dragged to set a new value.

Learn more about [Events](#).

5.2.5 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.2.6 Example

C

Simple Arc

code

```
#include "../../lv_examples.h"

#if LV_USE_ARC

void lv_ex_arc_1(void)
{
    /*Create an Arc*/
    lv_obj_t * arc = lv_arc_create(lv_scr_act(), NULL);
    lv_arc_set_end_angle(arc, 200);
    lv_obj_set_size(arc, 150, 150);
    lv_obj_align(arc, NULL, LV_ALIGN_CENTER, 0, 0);
}

#endif
```

Loader with Arc

code

```
#include "../../lv_examples.h"
#if LV_USE_ARC

/**
 * An `lv_task` to call periodically to set the angles of the arc
 * @param t
 */
static void arc_loader(lv_task_t * t)
{
    static int16_t a = 270;

    a+=5;

    lv_arc_set_end_angle(t->user_data, a);

    if(a >= 270 + 360) {
        lv_task_del(t);
        return;
    }
}

/**
 * Create an arc which acts as a loader.
 */
void lv_ex_arc_2(void)
{
    /*Create an Arc*/
    lv_obj_t * arc = lv_arc_create(lv_scr_act(), NULL);
```

(continues on next page)

(continued from previous page)

```

lv_arc_set_bg_angles(arc, 0, 360);
lv_arc_set_angles(arc, 270, 270);
lv_obj_align(arc, NULL, LV_ALIGN_CENTER, 0, 0);

/* Create an `lv_task` to update the arc.
 * Store the `arc` in the user data*/
lv_task_create(arc_loader, 20, LV_TASK_PRIO_LOWEST, arc);
}

#endif

```

MicroPython

No examples yet.

5.3 Bar (lv_bar)

5.3.1 Overview

The bar object has a background and an indicator on it. The width of the indicator is set according to the current value of the bar.

Vertical bars can be created if the width of the object is smaller than its height.

Not only end, but the start value of the bar can be set which changes the start position of the indicator.

5.3.2 Parts and Styles

The Bar's main part is called **LV_BAR_PART_BG** and it uses the typical background style properties.

LV_BAR_PART_INDIC is a virtual part which also uses all the typical background properties. By default the indicator maximal size is the same as the background's size but setting positive padding values in **LV_BAR_PART_BG** will make the indicator smaller. (negative values will make it larger) If the *value* style property is used on the indicator the alignment will be calculated based on the current size of the indicator. For example a center aligned value is always shown in the middle of the indicator regardless it's current size.

5.3.3 Usage

Value and range

A new value can be set by **lv_bar_set_value(bar, new_value, LV_ANIM_ON/OFF)**. The value is interpreted in a 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.

It's also possible to set the start value of the bar using **lv_bar_set_start_value(bar, new_value, LV_ANIM_ON/OFF)**

Modes

The bar can be drawn symmetrical to zero (drawn from zero, left to right), if it's enabled with `lv_bar_set_type(bar, LV_BAR_TYPE_SYMMETRICAL)`.

5.3.4 Events

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

Learn more about [Events](#).

5.3.5 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.3.6 Example

C

Simple Bar

code

```
#include "../../lv_examples.h"
#if LV_USE_BAR

void lv_ex_bar_1(void)
{
    lv_obj_t * bar1 = lv_bar_create(lv_scr_act(), NULL);
    lv_obj_set_size(bar1, 200, 20);
    lv_obj_align(bar1, NULL, LV_ALIGN_CENTER, 0, 0);
    lv_bar_set_anim_time(bar1, 2000);
    lv_bar_set_value(bar1, 100, LV_ANIM_ON);
}

#endif
```

MicroPython

No examples yet.

5.4 Button (lv_btn)

5.4.1 Overview

Buttons are simple rectangle-like objects. They are derived from *Containers* so *layout* and *fit* are also available. Besides, it can be enabled to automatically go to checked state on click.

5.4.2 Parts and Styles

The buttons has only a main style called `LV_BTN_PART_MAIN` and it can use all the properties from the following groups:

- background
- border
- outline
- shadow
- value
- pattern
- transitions

It also uses the *padding* properties when *layout* or *fit* is enabled.

5.4.3 Usage

States

To make buttons usage simpler the button's state can be get with `lv_btn_get_state(btn)`. It returns one of the following values:

- `LV_BTN_STATE_RELEASED`
- `LV_BTN_STATE_PRESSED`
- `LV_BTN_STATE_CHECKED_RELEASED`
- `LV_BTN_STATE_CHECKED_PRESSED`
- `LV_BTN_STATE_DISABLED`
- `LV_BTN_STATE_CHECKED_DISABLED`

With `lv_btn_get_state(btn, LV_BTN_STATE_...)` the buttons state can be changed manually.

If a more precise description of the state is required (e.g. focused) the general `lv_obj_get_state(btn)` can be used.

Checkable

You can configure the buttons as *toggle button* with `lv_btn_set_checkable(btn, true)`. In this case, on click, the button goes to `LV_STATE_CHECKED` state automatically, or back when clicked again.

Layout and Fit

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.

5.4.4 Events

Besides the *Generic events* the following *Special events* are sent by the buttons:

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

Learn more about *Events*.

5.4.5 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, the state of `LV_KEY_ENTER` is translated to `LV_EVENT_PRESSED/PRESSING/RELEASED` etc.

Learn more about *Keys*.

5.4.6 Example

C

Simple Buttons

code

```
#include "../../lv_examples.h"
#include <stdio.h>
#if LV_USE_BTN

static void event_handler(lv_obj_t * obj, lv_event_t event)
{
    if(event == LV_EVENT_CLICKED) {
        printf("Clicked\n");
    }
}
```

(continues on next page)

(continued from previous page)

```

    else if(event == LV_EVENT_VALUE_CHANGED) {
        printf("Toggled\n");
    }
}

void lv_ex_btn_1(void)
{
    lv_obj_t * label;

    lv_obj_t * btn1 = lv_btn_create(lv_scr_act(), NULL);
    lv_obj_set_event_cb(btn1, event_handler);
    lv_obj_align(btn1, NULL, LV_ALIGN_CENTER, 0, -40);

    label = lv_label_create(btn1, NULL);
    lv_label_set_text(label, "Button");

    lv_obj_t * btn2 = lv_btn_create(lv_scr_act(), NULL);
    lv_obj_set_event_cb(btn2, event_handler);
    lv_obj_align(btn2, NULL, LV_ALIGN_CENTER, 0, 40);
    lv_btn_set_checkable(btn2, true);
    lv_btn_toggle(btn2);
    lv_btn_set_fit2(btn2, LV_FIT_NONE, LV_FIT_TIGHT);

    label = lv_label_create(btn2, NULL);
    lv_label_set_text(label, "Toggled");
}
#endif

```

code

```

#include "../../lv_examples.h"
#include <stdio.h>
#if LV_USE_BTN

/**
 * Advanced button transition examples
 */
void lv_ex_btn_2(void)
{
    static lv_anim_path_t path_overshoot;
    lv_anim_path_init(&path_overshoot);
    lv_anim_path_set_cb(&path_overshoot, lv_anim_path_overshoot);

    static lv_anim_path_t path_ease_out;
    lv_anim_path_init(&path_ease_out);
    lv_anim_path_set_cb(&path_ease_out, lv_anim_path_ease_out);

    static lv_anim_path_t path_ease_in_out;
    lv_anim_path_init(&path_ease_in_out);
    lv_anim_path_set_cb(&path_ease_in_out, lv_anim_path_ease_in_out);

    /*Gum-like button*/
    static lv_style_t style_gum;
    lv_style_init(&style_gum);
    lv_style_set_transform_width(&style_gum, LV_STATE_PRESSED, 10);
    lv_style_set_transform_height(&style_gum, LV_STATE_PRESSED, -10);
}

```

(continues on next page)

(continued from previous page)

```

lv_style_set_value_letter_space(&style_gum, LV_STATE_PRESSED, 5);
lv_style_set_transition_path(&style_gum, LV_STATE_DEFAULT, &path_overshoot);
lv_style_set_transition_path(&style_gum, LV_STATE_PRESSED, &path_ease_in_out);
lv_style_set_transition_time(&style_gum, LV_STATE_DEFAULT, 250);
lv_style_set_transition_delay(&style_gum, LV_STATE_DEFAULT, 100);
lv_style_set_transition_prop_1(&style_gum, LV_STATE_DEFAULT, LV_STYLE_TRANSFORM
↪ WIDTH);
lv_style_set_transition_prop_2(&style_gum, LV_STATE_DEFAULT, LV_STYLE_TRANSFORM
↪ HEIGHT);
lv_style_set_transition_prop_3(&style_gum, LV_STATE_DEFAULT, LV_STYLE_VALUE_
↪ LETTER_SPACE);

lv_obj_t * btn1 = lv_btn_create(lv_scr_act(), NULL);
lv_obj_align(btn1, NULL, LV_ALIGN_CENTER, 0, -80);
lv_obj_add_style(btn1, LV_BTN_PART_MAIN, &style_gum);

/*Instead of creating a label add a values string*/
lv_obj_set_style_local_value_str(btn1, LV_BTN_PART_MAIN, LV_STATE_DEFAULT, "Gum");

/*Halo on press*/
static lv_style_t style_halo;
lv_style_init(&style_halo);
lv_style_set_transition_time(&style_halo, LV_STATE_PRESSED, 400);
lv_style_set_transition_time(&style_halo, LV_STATE_DEFAULT, 0);
lv_style_set_transition_delay(&style_halo, LV_STATE_DEFAULT, 200);
lv_style_set_outline_width(&style_halo, LV_STATE_DEFAULT, 0);
lv_style_set_outline_width(&style_halo, LV_STATE_PRESSED, 20);
lv_style_set_outline_opa(&style_halo, LV_STATE_DEFAULT, LV_OPA_COVER);
lv_style_set_outline_opa(&style_halo, LV_STATE_FOCUSED, LV_OPA_COVER); /*Just_
↪ to be sure, the theme might use it*/
lv_style_set_outline_opa(&style_halo, LV_STATE_PRESSED, LV_OPA_TRANSP);
lv_style_set_transition_prop_1(&style_halo, LV_STATE_DEFAULT, LV_STYLE_OUTLINE_
↪ OPA);
lv_style_set_transition_prop_2(&style_halo, LV_STATE_DEFAULT, LV_STYLE_OUTLINE_
↪ WIDTH);

lv_obj_t * btn2 = lv_btn_create(lv_scr_act(), NULL);
lv_obj_align(btn2, NULL, LV_ALIGN_CENTER, 0, 0);
lv_obj_add_style(btn2, LV_BTN_PART_MAIN, &style_halo);
lv_obj_set_style_local_value_str(btn2, LV_BTN_PART_MAIN, LV_STATE_DEFAULT, "Halo
↪");

/*Ripple on press*/
static lv_style_t style_ripple;
lv_style_init(&style_ripple);
lv_style_set_transition_time(&style_ripple, LV_STATE_PRESSED, 300);
lv_style_set_transition_time(&style_ripple, LV_STATE_DEFAULT, 0);
lv_style_set_transition_delay(&style_ripple, LV_STATE_DEFAULT, 300);
lv_style_set_bg_opa(&style_ripple, LV_STATE_DEFAULT, 0);
lv_style_set_bg_opa(&style_ripple, LV_STATE_PRESSED, LV_OPA_80);
lv_style_set_border_width(&style_ripple, LV_STATE_DEFAULT, 0);
lv_style_set_outline_width(&style_ripple, LV_STATE_DEFAULT, 0);
lv_style_set_transform_width(&style_ripple, LV_STATE_DEFAULT, -20);
lv_style_set_transform_height(&style_ripple, LV_STATE_DEFAULT, -20);
lv_style_set_transform_width(&style_ripple, LV_STATE_PRESSED, 0);
lv_style_set_transform_height(&style_ripple, LV_STATE_PRESSED, 0);

```

(continues on next page)

(continued from previous page)

```

    lv_style_set_transition_path(&style_ripple, LV_STATE_DEFAULT, &path_ease_out);
    lv_style_set_transition_prop_1(&style_ripple, LV_STATE_DEFAULT, LV_STYLE_BG_OPA);
    lv_style_set_transition_prop_2(&style_ripple, LV_STATE_DEFAULT, LV_STYLE_
↪ TRANSFORM_WIDTH);
    lv_style_set_transition_prop_3(&style_ripple, LV_STATE_DEFAULT, LV_STYLE_
↪ TRANSFORM_HEIGHT);

    lv_obj_t * btn3 = lv_btn_create(lv_scr_act(), NULL);
    lv_obj_align(btn3, NULL, LV_ALIGN_CENTER, 0, 80);
    lv_obj_add_style(btn3, LV_BTN_PART_MAIN, &style_ripple);
    lv_obj_set_style_local_value_str(btn3, LV_BTN_PART_MAIN, LV_STATE_DEFAULT, "Ripple
↪");
}
#endif

```

MicroPython

No examples yet.

5.5 Button matrix (lv_btnmatrix)

5.5.1 Overview

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

The main reasons for wanting to use a button matrix instead of a container and individual button objects are:

- The button matrix is simpler to use for grid-based button layouts.
- The button matrix consumes a lot less memory per button.

5.5.2 Parts and Styles

The Button matrix's main part is called `LV_BTNMATRIX_PART_BG`. It draws a background using the typical background style properties.

`LV_BTNMATRIX_PART_BTN` is virtual part and it refers to the buttons on the button matrix. It also uses all the typical background properties.

The top/bottom/left/right padding values from the background are used to keep some space on the sides. Inner padding is applied between the buttons.

5.5.3 Usage

Button's text

There is a text on each button. To specify them a descriptor string array, called *map*, needs to be used. The map can be set with `lv_btnmatrix_set_map(btnm, my_map)`. The declaration of a map should look 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", ""}`. Each line's buttons have their width calculated automatically.

Control buttons

The **buttons width** can be set relative to the other button in the same line with `lv_btnmatrix_set_btn_width(btnm, btn_id, width)` E.g. in a line with two buttons: *btnA*, *width = 1* and *btnB*, *width = 2*, *btnA* will have 33 % width and *btnB* will have 66 % width. It's similar to how the **flex-grow** property works in CSS.

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

- **LV_BTNMATRIX_CTRL_HIDDEN** - make a button hidden (hidden buttons still take up space in the layout, they are just not visible or clickable)
- **LV_BTNMATRIX_CTRL_NO_REPEAT** - disable repeating when the button is long pressed
- **LV_BTNMATRIX_CTRL_DISABLED** - make a button disabled
- **LV_BTNMATRIX_CTRL_CHECKABLE** - enable toggling of a button
- **LV_BTNMATRIX_CTRL_CHECK_STATE** - set the toggle state
- **LV_BTNMATRIX_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_btnmatrix_set_btn_ctrl(btnm, btn_id, LV_BTNM_CTRL_...)` and `lv_btnmatrix_clear_btn_ctrl(btnm, btn_id, LV_BTNM_CTRL_...)` respectively. More `LV_BTNM_CTRL_...` values can be *Ored*

The set/clear the same control attribute for all buttons of a button matrix, use `lv_btnmatrix_set_btn_ctrl_all(btnm, btn_id, LV_BTNM_CTRL_...)` and `lv_btnmatrix_clear_btn_ctrl_all(btnm, btn_id, LV_BTNM_CTRL_...)`.

The set a control map for a button matrix (similarly to the map for the text), use `lv_btnmatrix_set_ctrl_map(btnm, ctrl_map)`. An element of `ctrl_map` should look like `ctrl_map[0] = width | LV_BTNM_CTRL_NO_REPEAT | LV_BTNM_CTRL_TGL_ENABLE`. The number of elements should be equal to the number of buttons (excluding newlines characters).

One check

The "One check" feature can be enabled with `lv_btnmatrix_set_one_check(btnm, true)` to allow only one button to be checked (toggled) at once.

Recolor

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

Aligning the button's text

To align the text on the buttons, use `lv_btnmatrix_set_align(roller, LV_LABEL_ALIGN_LEFT/CENTER/RIGHT)`. All text items in the button matrix will conform to the alignment property as it is set.

Notes

The Button matrix object is very light weighted because the buttons are not created just virtually drawn on the fly. This way, 1 button use only 8 extra bytes instead of the ~100-150 byte size of a normal [Button](#) object (plus the size of its container and a label for each button).

The disadvantage of this setup is that the ability to style individual buttons to be different from others is limited (aside from the toggling feature). If you require that ability, using individual buttons is very likely to be a better approach.

5.5.4 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 the ID of the pressed/released button.

Learn more about [Events](#).

5.5.5 Keys

The following *Keys* are processed by the Buttons:

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

Learn more about [Keys](#).

5.5.6 Example

C

Simple Button matrix

code

```
#include "../../lv_examples.h"
#include <stdio.h>
#if LV_USE_BTNMATRIX

static void event_handler(lv_obj_t * obj, lv_event_t event)
{
    if(event == LV_EVENT_VALUE_CHANGED) {
        const char * txt = lv_btnmatrix_get_active_btn_text(obj);

        printf("%s was pressed\n", txt);
    }
}

static const char * btnm_map[] = {"1", "2", "3", "4", "5", "\n",
                                   "6", "7", "8", "9", "0", "\n",
                                   "Action1", "Action2", ""};

void lv_ex_btnmatrix_1(void)
{
    lv_obj_t * btnm1 = lv_btnmatrix_create(lv_scr_act(), NULL);
    lv_btnmatrix_set_map(btnm1, btnm_map);
    lv_btnmatrix_set_btn_width(btnm1, 10, 2);           /*Make "Action1" twice as wide_
↪as "Action2"*/
    lv_btnmatrix_set_btn_ctrl(btnm1, 10, LV_BTNMATRIX_CTRL_CHECKABLE);
    lv_btnmatrix_set_btn_ctrl(btnm1, 11, LV_BTNMATRIX_CTRL_CHECK_STATE);
    lv_obj_align(btnm1, NULL, LV_ALIGN_CENTER, 0, 0);
    lv_obj_set_event_cb(btnm1, event_handler);
}

#endif
```

MicroPython

No examples yet.

5.6 Calendar (lv_calendar)

5.6.1 Overview

The Calendar object is a classic calendar which can:

- highlight the current day
- highlight any user-defined dates
- display the name of the days

- go the next/previous month by button click
- highlight the clicked day

5.6.2 Parts and Styles

The calendar's main part is called `LV_CALENDAR_PART_BG`. It draws a background using the typical background style properties.

Besides the following virtual parts exist:

- `LV_CALENDAR_PART_HEADER` The upper area where the current year and month's name is shown. It also has buttons to move the next/previous month. It uses typical background properties and padding to keep some distance from the background (top, left, right) and the day names (bottom).
- `LV_CALENDAR_PART_DAY_NAMES` Shows the name of the days below the header. It uses the *text* style properties padding to keep some distance from the background (left, right), header (top) and dates (bottom).
- `LV_CALENDAR_PART_DATES` Show the date numbers from 1..28/29/30/31 (depending on current month). Different "state" of the states are drawn according to the states defined in this part:
 - normal dates: drawn with `LV_STATE_DEFAULT` style
 - pressed date: drawn with `LV_STATE_PRESSED` style
 - today: drawn with `LV_STATE_FOCUSED` style
 - highlighted dates: drawn with `LV_STATE_CHECKED` style

5.6.3 Usage

5.6.4 Overview

To 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.

Current date

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

Shown date

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

Highlighted days

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

Name of the days

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", ...};`

Name of the months

Similarly to `day_names`, the name of the month can be set with `lv_calendar_set_month_names(calendar, month_names_array)`.

5.6.5 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.

5.6.6 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.6.7 Example

C

Calendar with day select

code

```
#include "../../lv_examples.h"
#include <stdio.h>

#if LV_USE_CALENDAR

static void event_handler(lv_obj_t * obj, lv_event_t event)
{
    if(event == LV_EVENT_VALUE_CHANGED) {
        lv_calendar_date_t * date = lv_calendar_get_pressed_date(obj);
        if(date) {
            printf("Clicked date: %02d.%02d.%d\n", date->day, date->month, date->
↪year);
        }
    }
}
```

(continues on next page)

(continued from previous page)

```

    }
}

void lv_ex_calendar_1(void)
{
    lv_obj_t * calendar = lv_calendar_create(lv_scr_act(), NULL);
    lv_obj_set_size(calendar, 235, 235);
    lv_obj_align(calendar, NULL, LV_ALIGN_CENTER, 0, 0);
    lv_obj_set_event_cb(calendar, event_handler);

    /*Make the date number smaller to be sure they fit into their area*/
    lv_obj_set_style_local_text_font(calendar, LV_CALENDAR_PART_DATE, LV_STATE_
↪DEFAULT, lv_theme_get_font_small());

    /*Set today's date*/
    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 a few days*/
    static lv_calendar_date_t highlighted_days[3];          /*Only its pointer will be_
↪saved so should be static*/
    highlighted_days[0].year = 2018;
    highlighted_days[0].month = 10;
    highlighted_days[0].day = 6;

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

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

    lv_calendar_set_highlighted_dates(calendar, highlighted_days, 3);
}

#endif

```

MicroPython

No examples yet.

5.7 Canvas (lv_canvas)

5.7.1 Overview

A Canvas inherits from *Image* where the user can draw anything. Rectangles, texts, images, lines arcs can be drawn here using lvgl's drawing engine. Besides some "effects" can be applied as well like rotation, zoom and blur.

5.7.2 Parts and Styles

The Canvas has on one main part called `LV_CANVAS_PART_MAIN` and only the *image_recolor* property is used to give a color to `LV_IMG_CF_ALPHA_1/2/4/8BIT` images.

5.7.3 Usage

Buffer

The Canvas needs a buffer which stores the drawn image. To assign a buffer to a Canvas, use `lv_canvas_set_buffer(canvas, buffer, width, height, LV_IMG_CF_...)`. Where *buffer* is a 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.

The canvas supports all the built-in color formats like `LV_IMG_CF_TRUE_COLOR` or `LV_IMG_CF_INDEXED_2BIT`. See the full list in the [Color formats](#) section.

Palette

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

Drawing

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;`

`lv_canvas_fill_bg(canvas, LV_COLOR_BLUE, LV_OPA_50)` fills the whole canvas to blue with 50% opacity. Note that, if the current color format doesn't support colors (e.g. `LV_IMG_CF_ALPHA_2BIT`) the color will be ignored. Similarly, if opacity is not supported (e.g. `LV_IMG_CF_TRUE_COLOR`) it will be ignored.

An array of pixels 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 to match.

To draw something to the canvas use

- `lv_canvas_draw_rect(canvas, x, y, width, height, &draw_dsc)`
- `lv_canvas_draw_text(canvas, x, y, max_width, &draw_dsc, txt, LV_LABEL_ALIGN_LEFT/CENTER/RIGHT)`
- `lv_canvas_draw_img(canvas, x, y, &img_src, &draw_dsc)`

- `lv_canvas_draw_line(canvas, point_array, point_cnt, &draw_dsc)`
- `lv_canvas_draw_polygon(canvas, points_array, point_cnt, &draw_dsc)`
- `lv_canvas_draw_arc(canvas, x, y, radius, start_angle, end_angle, &draw_dsc)`

`draw_dsc` is a `lv_draw_rect/label/img/line_dsc_t` variable which should be first initialized with `lv_draw_rect/label/img/line_dsc_init()` function and then it's field should be modified with the desired colors and other values.

The draw function can draw to any color format. For example, it's possible to draw a text to an `LV_IMG_VF_ALPHA_8BIT` canvas and use the result image as a mask in *lv_objmask* later.

Transformations

`lv_canvas_transform()` can be used to rotate and/or scale the image of an image and store the result on the canvas. The function needs the following parameters:

- **canvas** pointer to a canvas object to store the result of the transformation.
- **img pointer** to an image descriptor to transform. Can be the image descriptor of an other canvas too (`lv_canvas_get_img()`).
- **angle** the angle of rotation (0..3600), 0.1 deg resolution
- **zoom** zoom factor (256 no zoom, 512 double size, 128 half size);
- **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
- **antialias** true: apply anti-aliasing during the transformation. Looks better but slower.

Note that a canvas can't be rotated on itself. You need a source and destination canvas or image.

5.7.4 Blur

A given area of the canvas can be blurred horizontally with `lv_canvas_blur_hor(canvas, &area, r)` to vertically with `lv_canvas_blur_ver(canvas, &area, r)`. `r` is the radius of the blur (greater value means more intensive blurring). `area` is the area where the blur should be applied (interpreted relative to the canvas)

5.7.5 Events

As default the clicking of a canvas is disabled (inherited by *Image*) and therefore no events are generated.

If clicking is enabled (`lv_obj_set_click(canvas, true)`) only the *Generic events* are sent by the object type.

Learn more about *Events*.

5.7.6 Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

5.7.7 Example

C

Drawing on the Canvas and rotate

code

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS

#define CANVAS_WIDTH 200
#define CANVAS_HEIGHT 150

void lv_ex_canvas_1(void)
{
    lv_draw_rect_dsc_t rect_dsc;
    lv_draw_rect_dsc_init(&rect_dsc);
    rect_dsc.radius = 10;
    rect_dsc.bg_opa = LV_OPA_COVER;
    rect_dsc.bg_grad_dir = LV_GRAD_DIR_HOR;
    rect_dsc.bg_color = LV_COLOR_RED;
    rect_dsc.bg_grad_color = LV_COLOR_BLUE;
    rect_dsc.border_width = 2;
    rect_dsc.border_opa = LV_OPA_90;
    rect_dsc.border_color = LV_COLOR_WHITE;
    rect_dsc.shadow_width = 5;
    rect_dsc.shadow_ofs_x = 5;
    rect_dsc.shadow_ofs_y = 5;

    lv_draw_label_dsc_t label_dsc;
    lv_draw_label_dsc_init(&label_dsc);
    label_dsc.color = LV_COLOR_YELLOW;

    static lv_color_t cbuf[LV_CANVAS_BUF_SIZE_TRUE_COLOR(CANVAS_WIDTH, CANVAS_
↵HEIGHT)];

    lv_obj_t * canvas = lv_canvas_create(lv_scr_act(), NULL);
    lv_canvas_set_buffer(canvas, cbuf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_IMG_CF_TRUE_
↵COLOR);
```

(continues on next page)

(continued from previous page)

```

lv_obj_align(canvas, NULL, LV_ALIGN_CENTER, 0, 0);
lv_canvas_fill_bg(canvas, LV_COLOR_SILVER, LV_OPA_COVER);

lv_canvas_draw_rect(canvas, 70, 60, 100, 70, &rect_dsc);

lv_canvas_draw_text(canvas, 40, 20, 100, &label_dsc, "Some text on text canvas",
↪ LV_LABEL_ALIGN_LEFT);

    /* Test the rotation. It requires an other buffer where the original image is
↪ stored.
    * So copy the current image to buffer and rotate it to the canvas */
    static lv_color_t cbuf_tmp[CANVAS_WIDTH * CANVAS_HEIGHT];
    memcpy(cbuf_tmp, cbuf, sizeof(cbuf_tmp));
    lv_img_dsc_t img;
    img.data = (void *)cbuf_tmp;
    img.header.cf = LV_IMG_CF_TRUE_COLOR;
    img.header.w = CANVAS_WIDTH;
    img.header.h = CANVAS_HEIGHT;

    lv_canvas_fill_bg(canvas, LV_COLOR_SILVER, LV_OPA_COVER);
    lv_canvas_transform(canvas, &img, 30, LV_IMG_ZOOM_NONE, 0, 0, CANVAS_WIDTH / 2,
↪ CANVAS_HEIGHT / 2, true);
}

#endif

```

Transparent Canvas with chroma keying

code

```

#include "../../lv_examples.h"
#if LV_USE_CANVAS

#define CANVAS_WIDTH 50
#define CANVAS_HEIGHT 50

/**
 * Create a transparent canvas with Chroma keying and indexed color format (palette).
 */
void lv_ex_canvas_2(void)
{
    /*Create a button to better see the transparency*/
    lv_btn_create(lv_scr_act(), NULL);

    /*Create a buffer for the canvas*/
    static lv_color_t cbuf[LV_CANVAS_BUF_SIZE_INDEXED_1BIT(CANVAS_WIDTH, CANVAS_
↪ HEIGHT)];

    /*Create a canvas and initialize its the palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_scr_act(), NULL);
    lv_canvas_set_buffer(canvas, cbuf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_IMG_CF_INDEXED_
↪ 1BIT);
    lv_canvas_set_palette(canvas, 0, LV_COLOR_TRANSP);
    lv_canvas_set_palette(canvas, 1, LV_COLOR_RED);
}

```

(continues on next page)

(continued from previous page)

```

    /*Create colors with the indices of the palette*/
    lv_color_t c0;
    lv_color_t c1;

    c0.full = 0;
    c1.full = 1;

    /*Transparent background*/
    lv_canvas_fill_bg(canvas, c1, LV_OPA_TRANSP);

    /*Create hole on the canvas*/
    uint32_t x;
    uint32_t y;
    for( y = 10; y < 30; y++) {
        for( x = 5; x < 20; x++) {
            lv_canvas_set_px(canvas, x, y, c0);
        }
    }
}
#endif

```

MicroPython

No examples yet.

5.8 Checkbox (lv_cb)

5.8.1 Overview

The Checkbox objects are built from a *Button* background which contains an also Button *bullet* and a *Label* to realize a classical checkbox.

5.8.2 Parts and Styles

The Check box's main part is called **LV_CHECKBOX_PART_BG**. It's a container for a "bullet" and a text next to it. The background uses all the typical background style properties.

The bullet is real *lv_obj* object and can be referred with **LV_CHECKBOX_PART_BULLET**. The bullet automatically inherits the state of the background. So the background is pressed the bullet goes to pressed state as well. The bullet also uses all the typical background style properties.

There is not dedicated part for the label. Its styles can be set in the background's styles because the *text* styles properties are always inherited.

5.8.3 Usage

Text

The text can be modified by the `lv_checkbox_set_text(cb, "New text")` function. It will dynamically allocate the text.

To set a static text, use `lv_checkbox_set_static_text(cb, txt)`. This way, only a pointer of `txt` will be stored and it shouldn't be deallocated while the checkbox exists.

Check/Uncheck

You can manually check / un-check the Checkbox via `lv_checkbox_set_checked(cb, true/false)`. Setting `true` will check the checkbox and `false` will un-check the checkbox.

Disabled

To make the Checkbox disabled, use `lv_checkbox_set_disabled(cb, true)`.

Get/Set Checkbox State

You can get the current state of the Checkbox with the `lv_checkbox_get_state(cb)` function which returns the current state. You can set the current state of the Checkbox with the `lv_checkbox_set_state(cb, state)`. The available states as defined by the enum `lv_btn_state_t` are:

- `LV_BTN_STATE_RELEASED`
- `LV_BTN_STATE_PRESSED`
- `LV_BTN_STATE_DISABLED`
- `LV_BTN_STATE_CHECKED_RELEASED`
- `LV_BTN_STATE_CHECKED_PRESSED`
- `LV_BTN_STATE_CHECKED_DISABLED`

5.8.4 Events

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

- `LV_EVENT_VALUE_CHANGED` - sent when the checkbox is toggled.

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

Learn more about [Events](#).

5.8.5 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.

Learn more about *Keys*.

5.8.6 Example

C

Simple Checkbox

code

```
#include "../../lv_examples.h"
#include <stdio.h>
#if LV_USE_CHECKBOX

static void event_handler(lv_obj_t * obj, lv_event_t event)
{
    if(event == LV_EVENT_VALUE_CHANGED) {
        printf("State: %s\n", lv_checkbox_is_checked(obj) ? "Checked" : "Unchecked");
    }
}

void lv_ex_checkbox_1(void)
{
    lv_obj_t * cb = lv_checkbox_create(lv_scr_act(), NULL);
    lv_checkbox_set_text(cb, "I agree to terms and conditions.");
    lv_obj_align(cb, NULL, LV_ALIGN_CENTER, 0, 0);
    lv_obj_set_event_cb(cb, event_handler);
}

#endif
```

MicroPython

No examples yet.

5.9 Chart (lv_chart)

5.9.1 Overview

Charts are a basic object to visualize data points. They support *Line* charts (connect points with lines and/or draw points on them) and *Column* charts.

Charts also support division lines, 2 y axis, axis ticks, and texts on ticks.

5.9.2 Parts and Styles

The Chart's main part is called `LV_CHART_PART_BG` and it uses all the typical background properties. The *text* style properties determine the style of the axis texts and the *line* properties determine ticks' style. *Padding* values add some space on the sides thus it makes *series area* smaller. Padding also can be used to make space for axis texts and ticks.

The background of the series is called `LV_CHART_PART_SERIES_BG` and it's placed on the main background. The division lines, and series data is drawn on this part. Besides the typical background style properties the *line* style properties are used by the division lines. The *padding* values tells the space between the this part and the axis texts.

The style of the series can be referenced by `LV_CHART_PART_SERIES`. In case of column type the following properties are used:

- *radius*: radius of the bars
- *padding_inner*: space between the columns of the same x coordinate

In case of Line type these properties are used:

- *line properties* to describe the lines
- *size* radius of the points
- *bg_opa*: the overall opacity of the area below the lines
- *bg_main_stop*: % of *bg_opa* at the top to create an alpha fade (0: transparent at the top, 255: *bg_opa* at the top)
- *bg_grad_stop*: % of *bg_opa* at the bottom to create an alpha fade (0: transparent at the bottom, 255: *bg_opa* at the top)
- *bg_grad_dir*: should be `LV_GRAD_DIR_VER` to allow alpha fading with *bg_main_stop* and *bg_grad_stop*

5.9.3 Usage

Data series

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 if not using an external array, if an external array is assigned any internal points associated with the series are deallocated and the series points to the external array instead.

Series' type

The following **data display types** exist:

- **LV_CHART_TYPE_NONE** - Do not display any data. It can be used to hide a series.
- **LV_CHART_TYPE_LINE** - Draw lines between the points.
- **LV_CHART_TYPE_COLUMN** - Draw columns.

You can specify the display type with `lv_chart_set_type(chart, LV_CHART_TYPE_...)`. The types can be 'OR'ed (like **LV_CHART_TYPE_LINE**).

Modify the data

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 `lv_chart_set_point_id(chart, ser, value, id)` where `id` is the index of the point you wish to update.
3. Use the `lv_chart_set_next(chart, ser, value)`.
4. Initialize all points to a given value with: `lv_chart_init_points(chart, ser, value)`.
5. Set all points from an array with: `lv_chart_set_points(chart, ser, value_array)`.

Use **LV_CHART_POINT_DEF** as value to make the library skip drawing that point, column, or line segment.

Override default start point for series

If you wish a plot to start from a point other than the default which is `point[0]` of the series, you can set an alternative index with the function `lv_chart_set_x_start_point(chart, ser, id)` where `id` is the new index position to start plotting from.

Set an external data source

You can make the chart series update from an external data source by assigning it with the function: `lv_chart_set_ext_array(chart, ser, array, point_cnt)` where `array` is an external array of `lv_coord_t` with `point_cnt` elements. Note: you should call `lv_chart_refresh(chart)` after the external data source has been updated, to update the chart.

Get current chart information

There are four functions to get information about a chart:

1. `lv_chart_get_type(chart)` returns the current chart type.
2. `lv_chart_get_point_count(chart)` returns the current chart point count.
3. `lv_chart_get_x_start_point(ser)` returns the current plotting index for the specified series.
4. `lv_chart_get_point_id(chart, ser, id)` returns the value of the data at a particular index(`id`) for the specified series.

Update modes

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

- **LV_CHART_UPDATE_MODE_SHIFT** - Shift old data to the left and add the new one on the right.
- **LV_CHART_UPDATE_MODE_CIRCULAR** - Circularly add the new data (Like an ECG diagram).

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

Number of points

The number of points in the series can be modified by `lv_chart_set_point_count(chart, point_num)`. The default value is 10. Note: this also affects the number of points processed when an external buffer is assigned to a series.

Vertical range

You can specify the minimum and maximum values in y-direction 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.

Division lines

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.

Tick marks and labels

Ticks and labels can be added to the axis.

`lv_chart_set_x_tick_text(chart, list_of_values, num_tick_marks, LV_CHART_AXIS_...)` set the ticks and texts on x axis. `list_of_values` is a string with '\n' terminated text (expect the last) with text for the ticks. E.g. `const char * list_of_values = "first\nsec\nthird"`. `list_of_values` can be NULL. If `list_of_values` is set then `num_tick_marks` tells the number of ticks between two labels. If `list_of_values` is NULL then it specifies the total number of ticks.

Major tick lines are drawn where text is placed, and *minor tick lines* are drawn elsewhere. `lv_chart_set_x_tick_length(chart, major_tick_len, minor_tick_len)` sets the length of tick lines on the x-axis.

The same functions exists for the y axis too: `lv_chart_set_y_tick_text` and `lv_chart_set_y_tick_length`.

5.9.4 Events

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

Learn more about [Events](#).

5.9.5 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.9.6 Example

C

Line Chart

code

```
#include "../../lv_examples.h"
#if LV_USE_CHART

void lv_ex_chart_1(void)
{
    /*Create a chart*/
    lv_obj_t * chart;
    chart = lv_chart_create(lv_scr_act(), NULL);
    lv_obj_set_size(chart, 200, 150);
    lv_obj_align(chart, NULL, LV_ALIGN_CENTER, 0, 0);
    lv_chart_set_type(chart, LV_CHART_TYPE_LINE); /*Show lines and points too*/

    /*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 'ser1'*/
    lv_chart_set_next(chart, ser1, 10);
    lv_chart_set_next(chart, ser1, 10);
    lv_chart_set_next(chart, ser1, 10);
    lv_chart_set_next(chart, ser1, 10);
    lv_chart_set_next(chart, ser1, 10);
    lv_chart_set_next(chart, ser1, 10);
    lv_chart_set_next(chart, ser1, 10);
    lv_chart_set_next(chart, ser1, 30);
    lv_chart_set_next(chart, ser1, 70);
    lv_chart_set_next(chart, ser1, 90);

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

(continues on next page)

(continued from previous page)

```

    ser2->points[6] = 65;
    ser2->points[7] = 65;
    ser2->points[8] = 65;
    ser2->points[9] = 65;

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

#endif

```

code

```

#include "../../lv_examples.h"
#if LV_USE_CHART

/**
 * Add a faded area effect to the line chart
 */
void lv_ex_chart_2(void)
{
    /*Create a chart*/
    lv_obj_t * chart;
    chart = lv_chart_create(lv_scr_act(), NULL);
    lv_obj_set_size(chart, 200, 150);
    lv_obj_align(chart, NULL, LV_ALIGN_CENTER, 0, 0);
    lv_chart_set_type(chart, LV_CHART_TYPE_LINE); /*Show lines and points too*/

    /*Add a faded are effect*/
    lv_obj_set_style_local_bg_opa(chart, LV_CHART_PART_SERIES, LV_STATE_DEFAULT, LV_
↪OPA_50); /*Max. opa.*/
    lv_obj_set_style_local_bg_grad_dir(chart, LV_CHART_PART_SERIES, LV_STATE_DEFAULT,
↪LV_GRAD_DIR_VER);
    lv_obj_set_style_local_bg_main_stop(chart, LV_CHART_PART_SERIES, LV_STATE_DEFAULT,
↪255); /*Max opa on the top*/
    lv_obj_set_style_local_bg_grad_stop(chart, LV_CHART_PART_SERIES, LV_STATE_DEFAULT,
↪0); /*Transparent on the bottom*/

    /*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 'ser1'*/
    lv_chart_set_next(chart, ser1, 31);
    lv_chart_set_next(chart, ser1, 66);
    lv_chart_set_next(chart, ser1, 10);
    lv_chart_set_next(chart, ser1, 89);
    lv_chart_set_next(chart, ser1, 63);
    lv_chart_set_next(chart, ser1, 56);
    lv_chart_set_next(chart, ser1, 32);
    lv_chart_set_next(chart, ser1, 35);
    lv_chart_set_next(chart, ser1, 57);
    lv_chart_set_next(chart, ser1, 85);

    /*Directly set points on 'ser2'*/
    ser2->points[0] = 92;

```

(continues on next page)

(continued from previous page)

```

ser2->points[1] = 71;
ser2->points[2] = 61;
ser2->points[3] = 15;
ser2->points[4] = 21;
ser2->points[5] = 35;
ser2->points[6] = 35;
ser2->points[7] = 58;
ser2->points[8] = 31;
ser2->points[9] = 53;

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

#endif

```

MicroPython

No examples yet.

5.10 Container (lv_cont)

5.10.1 Overview

The containers are essentially a **basic object** with layout and automatic sizing features.

5.10.2 Parts and Styles

The containers has only a main style called **LV_CONT_PART_MAIN** and it can use all the typically background properties and padding for layout auto sizing.

5.10.3 Usage

Layout

You can apply a layout on the containers to automatically order their children. The layout spacing comes from the style's **pad** 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 **padd_inner** space between them.
- **LV_LAYOUT_COLUMN_LEFT** - Align children in a left-justified column. Keep **padd_left** space on the left, **pad_top** space on the top and **pad_inner** space between the children.
- **LV_LAYOUT_COLUMN_MID** - Align children in centered column. Keep **padd_top** space on the top and **pad_inner** space between the children.
- **LV_LAYOUT_COLUMN_RIGHT** - Align children in a right-justified column. Keep **padd_right** space on the right, **pad_top** space on the top and **pad_inner** space between the children.

- **LV_LAYOUT_ROW_TOP** - Align children in a top justified row. Keep **pad_left** space on the left, **pad_top** space on the top and **pad_inner** space between the children.
- **LV_LAYOUT_ROW_MID** - Align children in centered row. Keep **pad_left** space on the left and **pad_inner** space between the children.
- **LV_LAYOUT_ROW_BOTTOM** - Align children in a bottom justified row. Keep **pad_left** space on the left, **pad_bottom** space on the bottom and **pad_inner** space between the children.
- **LV_LAYOUT_PRETTY_TOP** - Put as many objects as possible in a row (with at least **pad_inner** space and **pad_left/right** space on the sides). Divide the space in each line equally between the children. If here are children with different height in a row align their top edge.
- **LV_LAYOUT_PRETTY_MID** - Same as **LV_LAYOUT_PRETTY_TOP** but if here are children with different height in a row align their middle line.
- **LV_LAYOUT_PRETTY_BOTTOM** - Same as **LV_LAYOUT_PRETTY_TOP** but if here are children with different height in a row align their bottom line.
- **LV_LAYOUT_GRID** - Similar to **LV_LAYOUT_PRETTY** but not divide horizontal space equally just let **pad_left/right** on the edges and **pad_inner** space between the elements.

Autofit

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

- **LV_FIT_NONE** - Do not change the size automatically.
- **LV_FIT_TIGHT** - Shrink-wrap the container around all of its children, while keeping **pad_top/bottom/left/right** space on the edges.
- **LV_FIT_PARENT** - Set the size to the parent's size minus **pad_top/bottom/left/right** (from the parent's style) space.
- **LV_FIT_MAX** - Use **LV_FIT_PARENT** while smaller than the parent and **LV_FIT_TIGHT** when larger. It will ensure that the container is, at minimum, the size of its parent.

To set the auto fit mode for all directions, use `lv_cont_set_fit(cont, LV_FIT_...)`. 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)`.

5.10.4 Events

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

Learn more about [Events](#).

5.10.5 Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

5.10.6 Example

C

Container with auto-fit

code

```
#include "../../lv_examples.h"
#if LV_USE_CONT

void lv_ex_cont_1(void)
{
    lv_obj_t * cont;

    cont = lv_cont_create(lv_scr_act(), NULL);
    lv_obj_set_auto_realign(cont, true);           /*Auto realign when the_
↪size changes*/
    lv_obj_align_origo(cont, NULL, LV_ALIGN_CENTER, 0, 0); /*This parametrs will be_
↪sued when realigned*/
    lv_cont_set_fit(cont, LV_FIT_TIGHT);
    lv_cont_set_layout(cont, LV_LAYOUT_COLUMN_MID);

    lv_obj_t * label;
    label = lv_label_create(cont, NULL);
    lv_label_set_text(label, "Short text");

    /*Refresh and pause here for a while to see how `fit` works*/
    uint32_t t;
    lv_refr_now(NULL);
    t = lv_tick_get();
    while(lv_tick_elaps(t) < 500);

    label = lv_label_create(cont, NULL);
    lv_label_set_text(label, "It is a long text");

    /*Wait here too*/
    lv_refr_now(NULL);
    t = lv_tick_get();
    while(lv_tick_elaps(t) < 500);

    label = lv_label_create(cont, NULL);
    lv_label_set_text(label, "Here is an even longer text");
}

#endif
```

MicroPython

No examples yet.

5.11 color picker (lv_cpicker)

5.11.1 Overview

As its name implies *Color picker* allows to select color. The Hue, Saturation and Value of the color can be selected after each other.

The widget has two forms: circle (disc) and rectangle.

In both forms, by long pressing the object, the color picker will change to the next parameter of the color (hue, saturation or value). Besides, double click will reset the current parameter.

5.11.2 Parts and Styles

The Color picker's main part is called `LV_CPICKER_PART_BG`. In circular form it uses *scale_width* to set the width of the circle and *pad_inner* for padding between the circle and the inner preview circle. In rectangle mode *radius* can be used to apply a radius on the rectangle.

The object has virtual part called `LV_CPICKER_PART_KNOB` which is rectangle (or circle) drawn on the current value. It uses all the rectangle like style properties and padding to make it larger than the width of the circle or rectangle background.

5.11.3 Usage

Type

The type of the Color picker can be changed with `lv_cpicker_set_type(cpicker, LV_CPICKER_TYPE_RECT/DISC)`

Set color

The color can be set manually with `lv_cpicker_set_hue/saturation/value(cpicker, x)` or all at once with `lv_cpicker_set_hsv(cpicker, hsv)` or `lv_cpicker_set_color(cpicker, rgb)`

Color mode

The current color mode can be manually selected with `lv_cpicker_set_color_mode(cpicker, LV_CPICKER_COLOR_MODE_HUE/SATURATION/VALUE)`.

The color mode can be fixed (do not change with long press) using `lv_cpicker_set_color_mode_fixed(cpicker, true)`

Knob color

`lv_cpicker_set_knob_colored(cpicker, true)` make the knob to automatically show the selected color as background color.

5.11.4 Events

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

Learn more about [Events](#).

5.11.5 Keys

- **LV_KEY_UP**, **LV_KEY_RIGHT** Increment the current parameter's value by 1
- **LV_KEY_DOWN**, **LV_KEY_LEFT** Decrement the current parameter's by 1
- **LV_KEY_ENTER** By long press the next mode will be shown. By double click the current parameter will be reset.

Learn more about [Keys](#).

5.11.6 Example

C

Disc color picker

code

```
#include "../../lv_examples.h"
#if LV_USE_CPICKER

void lv_ex_cpicker_1(void)
{
    lv_obj_t * cpicker;

    cpicker = lv_cpicker_create(lv_scr_act(), NULL);
    lv_obj_set_size(cpicker, 200, 200);
    lv_obj_align(cpicker, NULL, LV_ALIGN_CENTER, 0, 0);
}

#endif
```

MicroPython

No examples yet.

5.12 Drop-down list (lv_dropdown)

5.12.1 Overview

The drop-down list allows the user to select one value from a list.

The drop-down list is closed by default and displays a single value or a predefined text. When activated (by click on the drop-down list), a list is created from which the user may select one option. When the user selects a new value, the list is deleted.

5.12.2 Parts and Styles

The drop-down list's main part is called `LV_DROPDOWN_PART_MAIN` which is a simple *lv_obj* object. It uses all the typical background properties. *Pressed*, *Focused*, *Edited* etc. styles are also applied as usual.

The list, which is created when the main object is clicked, is an *Page*. Its background part can be referenced with `LV_DROPDOWN_PART_LIST` and uses all the typical background properties for the rectangle itself and text properties for the options. To adjust the space between the options use the *text_line_space* style property. Padding values can be used to make some space on the edges.

The scrollable part of the page is hidden and its styles are always empty (so transparent with no padding).

The scrollbar can be referenced with `LV_DROPDOWN_PART_SCRLBAR` and uses all the typical background properties.

The selected option can be referenced with `LV_DROPDOWN_PART_SELECTED` and uses all the typical background properties. It will be used in its default state to draw a rectangle on the selected option, and in pressed state to draw a rectangle on the being pressed option.

5.12.3 Usage

5.12.4 Overview

Set options

The options are passed to the drop-down list as a string with `lv_dropdown_set_options(dropdown, options)`. The options should be separated by `\n`. For example: `"First\nSecond\nThird"`. The string will be saved in the drop-down list, so it can be in local variable too.

The `lv_dropdown_add_option(dropdown, "New option", pos)` function inserts a new option to `pos` index.

To save memory the options can be set from a static(constant) string too with `lv_dropdown_set_static_options(dropdown, options)`. In this case the options string should be alive while the drop-down list exists and `lv_dropdown_add_option` can't be used.

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

Get selected option

To get the currently selected option, use `lv_dropdown_get_selected(dropdown)`. It will return the *index* of the selected option.

`lv_dropdown_get_selected_str(dropdown, buf, buf_size)` copies the name of the selected option to a `buf`.

Direction

The list can be created on any side. The default `LV_DROPDOWN_DOWN` can be modified by `lv_dropdown_set_dir(dropdown, LV_DROPDOWN_DIR_LEFT/RIGHT/UP/DOWN)` function.

If the list would be vertically out of the screen, it will align to the edge.

Symbol

A symbol (typically an arrow) can be added to the drop down list with `lv_dropdown_set_symbol(dropdown, LV_SYMBOL_...)`

If the direction of the drop-down list is `LV_DROPDOWN_DIR_LEFT` the symbol will be shown on the left, else on the right.

Maximum height

The maximum height of drop-down list can be set via `lv_dropdown_set_max_height(dropdown, height)`. By default it's set to 3/4 vertical resolution.

Show selected

The main part can either show the selected option or a static text. It can be controlled with `lv_dropdown_set_show_selected(sropdown, true/false)`.

The static text can be set with `lv_dropdown_set_text(dropdown, "Text")`. Only the pointer of the text is saved.

If you also don't want the selected option to be highlighted, a custom transparent style can be used for `LV_DROPDOWN_PART_SELECTED`.

Animation time

The drop-down list's open/close animation time is adjusted by `lv_dropdown_set_anim_time(ddlist, anim_time)`. Zero animation time means no animation.

Manually open/close

To manually open or close the drop-down list the `lv_dropdown_open/close(dropdown, LV_ANIM_ON/OFF)` function can be used.

5.12.5 Events

Besides the [Generic events](#), the following [Special events](#) are sent by the drop-down list:

- **LV_EVENT_VALUE_CHANGED** - Sent when the new option is selected.

Learn more about [Events](#).

5.12.6 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).

5.12.7 Example

C

Simple Drop down list

code

```
#include "../../lv_examples.h"
#include <stdio.h>
#if LV_USE_DROPDOWN

static void event_handler(lv_obj_t * obj, lv_event_t event)
{
    if(event == LV_EVENT_VALUE_CHANGED) {
        char buf[32];
        lv_dropdown_get_selected_str(obj, buf, sizeof(buf));
        printf("Option: %s\n", buf);
    }
}

void lv_ex_dropdown_1(void)
{
    /*Create a normal drop down list*/
    lv_obj_t * ddlist = lv_dropdown_create(lv_scr_act(), NULL);
    lv_dropdown_set_options(ddlist, "Apple\n"
                                   "Banana\n"
                                   "Orange\n");
}
```

(continues on next page)

(continued from previous page)

```

        "Melon\n"
        "Grape\n"
        "Raspberry");

    lv_obj_align(ddlist, NULL, LV_ALIGN_IN_TOP_MID, 0, 20);
    lv_obj_set_event_cb(ddlist, event_handler);
}

#endif

```

Drop "up" list

code

```

#include "../../lv_examples.h"
#include <stdio.h>
#if LV_USE_DROPDOWN

/**
 * Create a drop LEFT menu
 */
void lv_ex_dropdown_2(void)
{
    /*Create a drop down list*/
    lv_obj_t * ddlist = lv_dropdown_create(lv_scr_act(), NULL);
    lv_dropdown_set_options(ddlist, "Apple\n"
        "Banana\n"
        "Orange\n"
        "Melon\n"
        "Grape\n"
        "Raspberry");

    lv_dropdown_set_dir(ddlist, LV_DROPDOWN_DIR_LEFT);
    lv_dropdown_set_symbol(ddlist, NULL);
    lv_dropdown_set_show_selected(ddlist, false);
    lv_dropdown_set_text(ddlist, "Fruits");
    /*It will be called automatically when the size changes*/
    lv_obj_align(ddlist, NULL, LV_ALIGN_IN_TOP_RIGHT, 0, 20);

    /*Copy the drop LEFT list*/
    ddlist = lv_dropdown_create(lv_scr_act(), ddlist);
    lv_obj_align(ddlist, NULL, LV_ALIGN_IN_TOP_RIGHT, 0, 100);
}

#endif

```

MicroPython

No examples yet.

5.13 Gauge (lv_gauge)

5.13.1 Overview

The gauge is a meter with scale labels and one or more needles.

5.13.2 Parts and Styles

The Gauge's main part is called `LV_GAUGE_PART_MAIN`. It draws a background using the typical background style properties and "minor" scale lines using the *line* and *scale* style properties. It also uses the *text* properties to set the style of the scale labels. *pad_inner* is used to set space between the scale lines and the scale labels.

`LV_GAUGE_PART_MAJOR` is a virtual part which describes the major scale lines (where labels are added) using the *line* and *scale* style properties.

`LV_GAUGE_PART_NEEDLE` is also virtual part and it describes the needle(s) via the *line* style properties. The *size* and the typical background properties are used to describe a rectangle (or circle) in the pivot point of the needle(s). *pad_inner* is used to make the needle(s) smaller than the outer radius of the scale lines.

5.13.3 Usage

Set value and needles

The gauge can show more than one needle. 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.

Scale

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 scale of the Gauge can have offset. It can be adjusted with `lv_gauge_set_angle_offset(gauge, angle)`.

Range

The range of the gauge can be specified by `lv_gauge_set_range(gauge, min, max)`. The default range is 0..100.

Needle image

An images also can be used as needles. The image should point to the right (like `==>`). To set an image use `lv_gauge_set_needle_img(gauge1, &img, pivot_x, pivot_y)`. `pivot_x` and `pivot_y` are offset of the rotation center from the top left corner. Images will be recolored to the needle's color with `image_recolor_opa` (style property) intensity.

Critical value

To set a critical value, use `lv_gauge_set_critical_value(gauge, value)`. The scale color will be changed to `scale_end_color` after this value. The default critical value is 80.

5.13.4 Events

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

Learn more about [Events](#).

5.13.5 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.13.6 Example

C

Simple Gauge

code

```
#include "../../lv_examples.h"
#if LV_USE_GAUGE

void lv_ex_gauge_1(void)
{
    /*Describe the color for the needles*/
    static lv_color_t needle_colors[3];
    needle_colors[0] = LV_COLOR_BLUE;
    needle_colors[1] = LV_COLOR_ORANGE;
    needle_colors[2] = LV_COLOR_PURPLE;

    /*Create a gauge*/
    lv_obj_t * gauge1 = lv_gauge_create(lv_scr_act(), NULL);
}
```

(continues on next page)

(continued from previous page)

```

lv_gauge_set_needle_count(gauge1, 3, needle_colors);
lv_obj_set_size(gauge1, 200, 200);
lv_obj_align(gauge1, NULL, LV_ALIGN_CENTER, 0, 0);

/*Set the values*/
lv_gauge_set_value(gauge1, 0, 10);
lv_gauge_set_value(gauge1, 1, 20);
lv_gauge_set_value(gauge1, 2, 30);
}

#endif

```

code

```

#include "../../lv_examples.h"
#if LV_USE_GAUGE

void lv_ex_gauge_2(void)
{
    /*Describe the color for the needles*/
    static lv_color_t needle_colors[3];
    needle_colors[0] = LV_COLOR_BLUE;
    needle_colors[1] = LV_COLOR_ORANGE;
    needle_colors[2] = LV_COLOR_PURPLE;

    LV_IMG_DECLARE(img_hand);

    /*Create a gauge*/
    lv_obj_t * gauge1 = lv_gauge_create(lv_scr_act(), NULL);
    lv_gauge_set_needle_count(gauge1, 3, needle_colors);
    lv_obj_set_size(gauge1, 200, 200);
    lv_obj_align(gauge1, NULL, LV_ALIGN_CENTER, 0, 0);
    lv_gauge_set_needle_img(gauge1, &img_hand, 4, 4);

    /*Set the values*/
    lv_gauge_set_value(gauge1, 0, 10);
    lv_gauge_set_value(gauge1, 1, 20);
    lv_gauge_set_value(gauge1, 2, 30);
}

#endif

```

MicroPython

No examples yet.

5.14 Image (lv_img)

5.14.1 Overview

Images are the basic object to display from the flash (as arrays) or externally as files. Images can display symbols (`LV_SYMBOL_...`) too.

Using the [Image decoder interface](#) custom image formats can be supported as well.

5.14.2 Parts and Styles

The images has only a main part called `LV_IMG_PART_MAIN` which uses the typical background style properties to draw a background rectangle and the *image* properties. The padding values are used to make the background virtually larger. (It won't change the image's real size but the size modification is applied only during drawing)

5.14.3 Usage

Image source

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 LVGL'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 sourced 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 of light-weighted mono-color "letters" instead of real images. You can set symbol like `lv_img_set_src(img1, LV_SYMBOL_OK)`.

Label as an image

Images and labels are sometimes used to convey the same thing. For example, to describe what a button does. Therefore, images and labels are somewhat interchangeable. To handle these images can even display texts by using `LV_SYMBOL_DUMMY` as the prefix of the text. For example, `lv_img_set_src(img, LV_SYMBOL_DUMMY "Some text")`.

Transparency

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.

Palette and Alpha index

Besides *True color* (RGB) color format, the following formats 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.

Recolor

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.

Auto-size

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.

Mosaic

If the object size is greater than the image size in any directions, then the image will be repeated like a mosaic. It's a very useful 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.

Offset

With `lv_img_set_offset_x(img, x_ofs)` and `lv_img_set_offset_y(img, y_ofs)`, you can add some offset to the displayed image. It is useful if the object size is smaller than the image source size. Using the offset parameter a [Texture atlas](#) or a "running image" effect can be created by [Animating](#) the x or y offset.

5.14.4 Transformations

Using the `lv_img_set_zoom(img, factor)` the images will be zoomed. Set `factor` to `256` or `LV_IMG_ZOOM_NONE` to disable zooming. A larger value enlarges the images (e.g. `512` double size), a smaller value shrinks it (e.g. `128` half size). Fractional scale works as well. E.g. `281` for 10% enlargement.

To rotate the image use `lv_img_set_angle(img, angle)`. Angle has 0.1 degree precision, so for 45.8° set `458`.

By default, the pivot point of the rotation is the center of the image. It can be changed with `lv_img_set_pivot(img, pivot_x, pivot_y)`. `0;0` is the top left corner.

The quality of the transformation can be adjusted with `lv_img_set_antialias(img, true/false)`. With enabled anti-aliasing the transformations has a higher quality but they are slower.

The transformations require the whole image to be available. Therefore indexed images (`LV_IMG_CF_INDEXED...`), alpha only images (`LV_IMG_CF_ALPHA...`) or images from files can be transformed. In other words transformations work only on true color images stored as C array, or if a custom [Image decoder](#) returns the whole image.

Note that, the real coordinates of image object won't change during transformation. That is `lv_obj_get_width/height/x/y()` will returned the original, non-zoomed coordinates.

5.14.5 Rotate

The images can be rotated with

5.14.6 Events

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

Learn more about [Events](#).

5.14.7 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.14.8 Example

C

Image from variable and symbol

code

```
#include "../../lv_examples.h"
#if LV_USE_IMG

/* Find the image here: https://github.com/lvgl/lv\_examples/tree/master/assets */
LV_IMG_DECLARE(img_cogwheel_argb);

void lv_ex_img_1(void)
{
    lv_obj_t * img1 = lv_img_create(lv_scr_act(), NULL);
    lv_img_set_src(img1, &img_cogwheel_argb);
    lv_obj_align(img1, NULL, LV_ALIGN_CENTER, 0, -20);

    lv_obj_t * img2 = lv_img_create(lv_scr_act(), NULL);
    lv_img_set_src(img2, LV_SYMBOL_OK "Accept");
    lv_obj_align(img2, img1, LV_ALIGN_OUT_BOTTOM_MID, 0, 20);
}

#endif
```

Image recoloring

code

```
#include "../../lv_examples.h"
#if LV_USE_IMG

#define SLIDER_WIDTH 20

static void create_sliders(void);
static void slider_event_cb(lv_obj_t * slider, lv_event_t event);

static lv_obj_t * red_slider, * green_slider, * blue_slider, * intense_slider;
static lv_obj_t * img1;
LV_IMG_DECLARE(img_cogwheel_argb);

void lv_ex_img_2(void)
{
    /*Create 4 sliders to adjust RGB color and re-color intensity*/
    create_sliders();

    /* Now create the actual image */
    img1 = lv_img_create(lv_scr_act(), NULL);
    lv_img_set_src(img1, &img_cogwheel_argb);
    lv_obj_align(img1, NULL, LV_ALIGN_IN_RIGHT_MID, -20, 0);
}

static void slider_event_cb(lv_obj_t * slider, lv_event_t event)
```

(continues on next page)

(continued from previous page)

```

{
    if(event == LV_EVENT_VALUE_CHANGED) {
        /* Recolor the image based on the sliders' values */
        lv_color_t color = lv_color_make(lv_slider_get_value(red_slider), lv_slider_
↪get_value(green_slider), lv_slider_get_value(blue_slider));
        lv_opa_t intense = lv_slider_get_value(intense_slider);
        lv_obj_set_style_local_image_recolor_opa(img1, LV_IMG_PART_MAIN, LV_STATE_
↪DEFAULT, intense);
        lv_obj_set_style_local_image_recolor(img1, LV_IMG_PART_MAIN, LV_STATE_DEFAULT,
↪ color);
    }
}

static void create_sliders(void)
{
    /* Create a set of RGB sliders */
    /* Use the red one as a base for all the settings */
    red_slider = lv_slider_create(lv_scr_act(), NULL);
    lv_slider_set_range(red_slider, 0, 255);
    lv_obj_set_size(red_slider, SLIDER_WIDTH, 200); /* Be sure it's a vertical slider ↪
↪*/
    lv_obj_set_style_local_bg_color(red_slider, LV_SLIDER_PART_INDIC, LV_STATE_
↪DEFAULT, LV_COLOR_RED);
    lv_obj_set_event_cb(red_slider, slider_event_cb);

    /* Copy it for the other three sliders */
    green_slider = lv_slider_create(lv_scr_act(), red_slider);
    lv_obj_set_style_local_bg_color(green_slider, LV_SLIDER_PART_INDIC, LV_STATE_
↪DEFAULT, LV_COLOR_LIME);

    blue_slider = lv_slider_create(lv_scr_act(), red_slider);
    lv_obj_set_style_local_bg_color(blue_slider, LV_SLIDER_PART_INDIC, LV_STATE_
↪DEFAULT, LV_COLOR_BLUE);

    intense_slider = lv_slider_create(lv_scr_act(), red_slider);
    lv_obj_set_style_local_bg_color(intense_slider, LV_SLIDER_PART_INDIC, LV_STATE_
↪DEFAULT, LV_COLOR_GRAY);
    lv_slider_set_value(intense_slider, 255, LV_ANIM_OFF);

    lv_obj_align(red_slider, NULL, LV_ALIGN_IN_LEFT_MID, 20, 0);
    lv_obj_align(green_slider, red_slider, LV_ALIGN_OUT_RIGHT_MID, 20, 0);
    lv_obj_align(blue_slider, green_slider, LV_ALIGN_OUT_RIGHT_MID, 20, 0);
    lv_obj_align(intense_slider, blue_slider, LV_ALIGN_OUT_RIGHT_MID, 20, 0);
}

#endif

```

MicroPython

No examples yet.

5.15 Image button (lv_imgbtn)

5.15.1 Overview

The Image button is very similar to the simple 'Button' object. The only difference is that, it displays user-defined images in each state instead of drawing a rectangle. Before reading this section, please read the [Button](#) section for better understanding.

5.15.2 Parts and Styles

The Image button object has only a main part called `LV_IMG_BTN_PART_MAIN` from where all *image* style properties are used. It's possible to recolor the image in each state with *image_recolor* and *image_recolor_opa* properties. For example, to make the image darker if it is pressed.

5.15.3 Usage

Image sources

To set the image in a state, use the `lv_imgbtn_set_src(imgbtn, LV_BTN_STATE_..., &img_src)`. The image sources works the same as described in the [Image object](#) except that, "Symbols" are not supported by the Image button.

If `LV_IMGBTN_TILED` is enabled in *lv_conf.h*, then `lv_imgbtn_set_src_tiled(imgbtn, LV_BTN_STATE_..., &img_src_left, &img_src_mid, &img_src_right)` becomes available. Using the tiled feature the *middle* image will be repeated to fill the width of the object. Therefore with `LV_IMGBTN_TILED`, you can set the width of the Image button using `lv_obj_set_width()`. However, without this option, the width will be always the same as the image source's width.

Button features

Similarly to normal Buttons `lv_imgbtn_set_checkable(imgbtn, true/false)`, `lv_imgbtn_toggle(imgbtn)` and `lv_imgbtn_set_state(imgbtn, LV_BTN_STATE_...)` also works.

5.15.4 Events

Beside 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 the 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](#).

5.15.5 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.

Learn more about *Keys*.

5.15.6 Example

C

Simple Image button

code

```
#include "../../lv_examples.h"
#if LV_USE_IMGBTN

void lv_ex_imgbtn_1(void)
{
    LV_IMG_DECLARE(imgbtn_green);
    LV_IMG_DECLARE(imgbtn_blue);

    /*Darken the button when pressed*/
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_image_recolor_opa(&style, LV_STATE_PRESSED, LV_OPA_30);
    lv_style_set_image_recolor(&style, LV_STATE_PRESSED, LV_COLOR_BLACK);
    lv_style_set_text_color(&style, LV_STATE_DEFAULT, LV_COLOR_WHITE);

    /*Create an Image button*/
    lv_obj_t * imgbtn1 = lv_imgbtn_create(lv_scr_act(), NULL);
    lv_imgbtn_set_src(imgbtn1, LV_BTN_STATE_RELEASED, &imgbtn_green);
    lv_imgbtn_set_src(imgbtn1, LV_BTN_STATE_PRESSED, &imgbtn_green);
    lv_imgbtn_set_src(imgbtn1, LV_BTN_STATE_CHECKED_RELEASED, &imgbtn_blue);
    lv_imgbtn_set_src(imgbtn1, LV_BTN_STATE_CHECKED_PRESSED, &imgbtn_blue);
    lv_imgbtn_set_checkable(imgbtn1, true);
    lv_obj_add_style(imgbtn1, LV_IMGBTN_PART_MAIN, &style);
    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");
}

#endif
```

MicroPython

No examples yet.

5.16 Keyboard (lv_keyboard)

5.16.1 Overview

The Keyboard object is a special *Button matrix* with predefined keymaps and other features to realize a virtual keyboard to write text.

5.16.2 Parts and Styles

Similarly to Button matrices Keyboards consist of 2 part:

- `LV_KEYBOARD_PART_BG` which is the main part and uses all the typical background properties
- `LV_KEYBOARD_PART_BTN` which is virtual part for the buttons. It also uses all typical background properties and the *text* properties.

5.16.3 Usage

Modes

The Keyboards have the following modes:

- `LV_KEYBOARD_MODE_TEXT_LOWER` - Display lower case letters
- `LV_KEYBOARD_MODE_TEXT_UPPER` - Display upper case letters
- `LV_KEYBOARD_MODE_TEXT_SPECIAL` - Display special characters
- `LV_KEYBOARD_MODE_NUM` - Display numbers, +/- sign, and decimal dot.

The TEXT modes' layout contains buttons to change mode.

To set the mode manually, use `lv_keyboard_set_mode(kb, mode)`. The default mode is `LV_KEYBOARD_MODE_TEXT_UPPER`.

Assign Text area

You can assign a *Text area* to the Keyboard to automatically put the clicked characters there. To assign the text area, use `lv_keyboard_set_textarea(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 one 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_keyboard_set_cursor_manage(kb, true)`. The default is not managed.

New Keymap

You can specify a new map (layout) for the keyboard with `lv_keyboard_set_map(kb, map)` and `lv_keyboard_set_ctrl_map(kb, ctrl_map)`. Learn more about the [Button matrix](#) object. Keep in mind that, using following keywords will have the same effect as with the original map:

- `LV_SYMBOL_OK` - Apply.
- `LV_SYMBOL_CLOSE` - Close.
- `LV_SYMBOL_BACKSPACE` - Delete on the left.
- `LV_SYMBOL_LEFT` - Move the cursor left.
- `LV_SYMBOL_RIGHT` - Move the cursor right.
- `"ABC"` - Load the uppercase map.
- `"abc"` - Load the lower case map.
- `"Enter"` - New line.

5.16.4 Events

Besides 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 the ID of the pressed/released button.
- `LV_EVENT_APPLY` - The *Ok* button is clicked.
- `LV_EVENT_CANCEL` - The *Close* button is clicked.

The keyboard has a **default event handler** callback called `lv_keyboard_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 however, you can call `lv_keyboard_def_event_cb` at the beginning of your event handler to handle the same things as before.

Learn more about [Events](#).

5.16.5 Keys

The following *Keys* are processed by the buttons:

- `LV_KEY_RIGHT/UP/LEFT/RIGHT` - To navigate among the buttons and select one.
- `LV_KEY_ENTER` - To press/release the selected button.

Learn more about [Keys](#).

5.16.6 Examples

C

Keyboard with text area

code

```
#include "../../lv_examples.h"
#if LV_USE_KEYBOARD

void lv_ex_keyboard_1(void)
{
    /*Create a keyboard and apply the styles*/
    lv_obj_t *kb = lv_keyboard_create(lv_scr_act(), NULL);
    lv_keyboard_set_cursor_manage(kb, true);

    /*Create a text area. The keyboard will write here*/
    lv_obj_t *ta = lv_textarea_create(lv_scr_act(), NULL);
    lv_obj_align(ta, NULL, LV_ALIGN_IN_TOP_MID, 0, LV_DPI / 16);
    lv_textarea_set_text(ta, "");
    lv_coord_t max_h = LV_VER_RES / 2 - LV_DPI / 8;
    if(lv_obj_get_height(ta) > max_h) lv_obj_set_height(ta, max_h);

    /*Assign the text area to the keyboard*/
    lv_keyboard_set_textarea(kb, ta);
}
#endif
```

MicroPython

Keyboard with text area

No examples yet.

5.17 Label (lv_label)

5.17.1 Overview

A label is the basic object type that is used to display text.

5.17.2 Parts and Styles

The label has only a main part, called **LV_LABEL_PART_MAIN**. It uses all the typical background properties and the *text* properties. The padding values can be used to make the area for the text small in the related direction.

5.17.3 Usage

Set text

You can set the text on a label at runtime with `lv_label_set_text(label, "New text")`. It will allocate a buffer dynamically, and the provided string will be copied into that buffer. Therefore, you don't need to keep the text you pass to `lv_label_set_text` in scope after that function returns.

With `lv_label_set_text_fmt(label, "Value: %d", 15)` **printf formatting** can be used to set the text.

Labels are able to show text from a **static character buffer** which is `\0`-terminated. To do so, use `lv_label_set_static_text(label, "Text")`. In this case, the text is not stored in the dynamic memory and the given buffer is used directly instead. This means that the array can't be a local variable which goes out of scope when the function exits. Constant strings are safe to use with `lv_label_set_static_text` (except when used with `LV_LABEL_LONG_DOT`, as it modifies the buffer in-place), as they are stored in ROM memory, which is always accessible.

You can also use a **raw 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 like with `lv_label_set_text`. To set a raw character array, use the `lv_label_set_array_text(label, char_array, size)` function.

Line break

Line breaks are handled automatically by the label object. You can use `\n` to make a line break. For example: `"line1\nline2\n\nline4"`

Long modes

By default, the width of the label object automatically expands to the text size. Otherwise, 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_DOT** - Keep the object size, break the text and write dots in the last line (not supported when using `lv_label_set_static_text`)
- **LV_LABEL_LONG_SCROLL** - Keep the size and scroll the label back and forth
- **LV_LABEL_LONG_SCROLL_CIRC** - Keep the size and scroll the label circularly
- **LV_LABEL_LONG_CROP** - Keep the size and crop the text out of it

You can specify the long mode with `lv_label_set_long_mode(label, LV_LABEL_LONG_...)`

It's important to note that, when a label is created and its text is set, the label's size already expanded to the text size. In addition with the default `LV_LABEL_LONG_EXPAND`, *long mode* `lv_obj_set_width/height/size()` has no effect.

So you need to change the *long mode* first set the new *long mode* and then set the size with `lv_obj_set_width/height/size()`.

Another important note is that **LV_LABEL_LONG_DOT** manipulates the text buffer in-place in order to add/remove the dots. When `lv_label_set_text` or `lv_label_set_array_text` are used, a separate buffer is allocated and this implementation detail is unnoticed. This is not the case with

`lv_label_set_static_text`! The buffer you pass to `lv_label_set_static_text` must be writable if you plan to use `LV_LABEL_LONG_DOT`.

Text align

The lines of the text can be aligned to the left, right or center with `lv_label_set_align(label, LV_LABEL_ALIGN_LEFT/RIGHT/CENTER)`. Note that, it will align only the lines, not the label object itself.

Vertical alignment is not supported by the label itself; you should place the label inside a larger container and align the whole label object instead.

Text recolor

In the text, you can use commands to recolor 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.

Note that, recoloring work only in a single line. Therefore, `\n` should not use in a recolored text or it should be wrapped by `LV_LABEL_LONG_BREAK` else, the text in the new line won't be recolored.

Very long texts

Lvgl can efficiently handle very long (> 40k characters) by saving some extra data (~12 bytes) to speed up drawing. To enable this feature, set `LV_LABEL_LONG_TXT_HINT 1` in `lv_conf.h`.

Symbols

The labels can display symbols alongside letters (or on their own). Read the [Font](#) section to learn more about the symbols.

5.17.4 Events

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

Learn more about [Events](#).

5.17.5 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.17.6 Example

C

Label recoloring and scrolling

code

```
#include "../../lv_examples.h"
#if LV_USE_LABEL

void lv_ex_label_1(void)
{
    lv_obj_t * label1 = lv_label_create(lv_scr_act(), NULL);
    lv_label_set_long_mode(label1, LV_LABEL_LONG_BREAK);      /*Break the long lines*/
    lv_label_set_recolor(label1, true);                        /*Enable re-coloring by
↳ commands in the text*/
    lv_label_set_align(label1, LV_LABEL_ALIGN_CENTER);         /*Center aligned lines*/
    lv_label_set_text(label1, "#0000ff Re-color# #ff00ff words# #ff0000 of a# label "
        "and wrap long text automatically.");
    lv_obj_set_width(label1, 150);
    lv_obj_align(label1, NULL, LV_ALIGN_CENTER, 0, -30);

    lv_obj_t * label2 = lv_label_create(lv_scr_act(), NULL);
    lv_label_set_long_mode(label2, LV_LABEL_LONG_SCROLL_CIRC); /*Circular scroll*/
    lv_obj_set_width(label2, 150);
    lv_label_set_text(label2, "It is a circularly scrolling text. ");
    lv_obj_align(label2, NULL, LV_ALIGN_CENTER, 0, 30);
}

#endif
```

Text shadow

code

```
#include "../../lv_examples.h"
#if LV_USE_LABEL

void lv_ex_label_2(void)
{
    /* Create a style for the shadow*/
    static lv_style_t label_shadow_style;
    lv_style_init(&label_shadow_style);
    lv_style_set_text_opa(&label_shadow_style, LV_STATE_DEFAULT, LV_OPA_50);
    lv_style_set_text_color(&label_shadow_style, LV_STATE_DEFAULT, LV_COLOR_RED);

    /*Create a label for the shadow first (it's in the background) */
    lv_obj_t * shadow_label = lv_label_create(lv_scr_act(), NULL);
    lv_obj_add_style(shadow_label, LV_LABEL_PART_MAIN, &label_shadow_style);

    /* Create the main label */
    lv_obj_t * main_label = lv_label_create(lv_scr_act(), NULL);
    lv_label_set_text(main_label, "A simple method to create\n"
        "shadows on text\n");
}
```

(continues on next page)

(continued from previous page)

```

        "It even works with\n\n"
        "newlines      and spaces.");

/*Set the same text for the shadow label*/
lv_label_set_text(shadow_label, lv_label_get_text(main_label));

/* Position the main label */
lv_obj_align(main_label, NULL, LV_ALIGN_CENTER, 0, 0);

/* Shift the second label down and to the right by 2 pixel */
lv_obj_align(shadow_label, main_label, LV_ALIGN_IN_TOP_LEFT, 1, 1);
}

#endif

```

Align labels

code

```

#include "../../lv_examples.h"
#if LV_USE_LABEL

static void text_changer(lv_task_t * t);

lv_obj_t * labels[3];

/**
 * Create three labels to demonstrate the alignments.
 */
void lv_ex_label_3(void)
{
    /*`lv_label_set_align` is not required to align the object itslef.
     * It's used only when the text has multiple lines*/

    /* Create a label on the top.
     * No additional alignment so it will be the reference*/
    labels[0] = lv_label_create(lv_scr_act(), NULL);
    lv_obj_align(labels[0], NULL, LV_ALIGN_IN_TOP_MID, 0, 5);
    lv_label_set_align(labels[0], LV_LABEL_ALIGN_CENTER);

    /* Create a label in the middle.
     * `lv_obj_align` will be called every time the text changes
     * to keep the middle position */
    labels[1] = lv_label_create(lv_scr_act(), NULL);
    lv_obj_align(labels[1], NULL, LV_ALIGN_CENTER, 0, 0);
    lv_label_set_align(labels[1], LV_LABEL_ALIGN_CENTER);

    /* Create a label in the bottom.
     * Enable auto realign. */
    labels[2] = lv_label_create(lv_scr_act(), NULL);
    lv_obj_set_auto_realign(labels[2], true);
    lv_obj_align(labels[2], NULL, LV_ALIGN_IN_BOTTOM_MID, 0, -5);
    lv_label_set_align(labels[2], LV_LABEL_ALIGN_CENTER);

    lv_task_t * t = lv_task_create(text_changer, 1000, LV_TASK_PRIO_MID, NULL);
}

```

(continues on next page)

(continued from previous page)

```

    lv_task_ready(t);
}

static void text_changer(lv_task_t * t)
{
    const char * texts[] = {"Text", "A very long text", "A text with\nmultiple\nlines
↪", NULL};
    static uint8_t i = 0;

    lv_label_set_text(labels[0], texts[i]);
    lv_label_set_text(labels[1], texts[i]);
    lv_label_set_text(labels[2], texts[i]);

    /*Manually realign `labels[1]`*/
    lv_obj_align(labels[1], NULL, LV_ALIGN_CENTER, 0, 0);

    i++;
    if(texts[i] == NULL) i = 0;
}

#endif

```

MicroPython

No examples yet.

5.18 LED (lv_led)

5.18.1 Overview

The LEDs are rectangle-like (or circle) object. It's brightness can be adjusted. With lower brightness the colors of the LED become darker.

5.18.2 Parts and Styles

The LEDs have only one main part, called `LV_LED_PART_MAIN` and it uses all the typical background style properties.

5.18.3 Usage

Brightness

You can set their brightness with `lv_led_set_bright(led, bright)`. The brightness should be between 0 (darkest) and 255 (lightest).

Toggle

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.

5.18.4 Events

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

Learn more about [Events](#).

5.18.5 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.18.6 Example

C

LED with custom style

code

```
#include "../../lv_examples.h"
#if LV_USE_LED

void lv_ex_led_1(void)
{
    /*Create a LED and switch it OFF*/
    lv_obj_t * led1 = lv_led_create(lv_scr_act(), NULL);
    lv_obj_align(led1, NULL, LV_ALIGN_CENTER, -80, 0);
    lv_led_off(led1);

    /*Copy the previous LED and set a brightness*/
    lv_obj_t * led2 = lv_led_create(lv_scr_act(), led1);
    lv_obj_align(led2, NULL, LV_ALIGN_CENTER, 0, 0);
    lv_led_set_bright(led2, 190);

    /*Copy the previous LED and switch it ON*/
    lv_obj_t * led3 = lv_led_create(lv_scr_act(), led1);
    lv_obj_align(led3, NULL, LV_ALIGN_CENTER, 80, 0);
    lv_led_on(led3);
}

#endif
```

MicroPython

No examples yet.

5.19 Line (lv_line)

5.19.1 Overview

The Line object is capable of drawing straight lines between a set of points.

5.19.2 Parts and Styles

The Line has only a main part, called `LV_LABEL_PART_MAIN`. It uses all the *line* style properties.

5.19.3 Usage

Set 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_cnt)` function.

Auto-size

It is possible to automatically set the size of the line object according to its points. It can be enable with the `lv_line_set_auto_size(line, true)` function. If enabled then when the points are set the object's width and height will be changed according to the maximal x and y coordinates among the points. The *auto size* is enabled by default.

Invert y

By default, the $y == 0$ point is in the top of the object. It might be conter-intuitive in some cases so the y coordinates can be inverted with `lv_line_set_y_invert(line, true)`. In this case, $y == 0$ will be the bottom of teh obejct. The *y invert* is disabled by default.

5.19.4 Events

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

Learn more about [Events](#).

5.19.5 Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

5.19.6 Example

C

Simple Line

code

```
#include "../../lv_examples.h"
#if LV_USE_LINE

void lv_ex_line_1(void)
{
    /*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 style*/
    static lv_style_t style_line;
    lv_style_init(&style_line);
    lv_style_set_line_width(&style_line, LV_STATE_DEFAULT, 8);
    lv_style_set_line_color(&style_line, LV_STATE_DEFAULT, LV_COLOR_BLUE);
    lv_style_set_line_rounded(&style_line, LV_STATE_DEFAULT, true);

    /*Create a line and apply the new 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_add_style(line1, LV_LINE_PART_MAIN, &style_line); /*Set the points*/
    lv_obj_align(line1, NULL, LV_ALIGN_CENTER, 0, 0);
}

#endif
```


MicroPython

No examples yet.

5.20 List (lv_list)

5.20.1 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 becomes long enough it can be scrolled.

5.20.2 Parts and Styles

The List has the same parts as the *Page*

- LV_LIST_PART_BG
- LV_LIST_PART_SCRL
- LV_LIST_PART_SCRLBAR
- LV_LIST_PART_EDGE_FLASH

Refer to the *Page* documentation for details.

The buttons on the list are treated as normal buttons and they only have a main part called LV_BTN_PART_MAIN.

5.20.3 Usage

Add buttons

You can add new list elements (button) with `lv_list_add_btn(list, &icon_img, "Text")` or with symbol `lv_list_add_btn(list, SYMBOL_EDIT, "Edit text")`. If you do not want to add image use `NULL` as image source. The function returns with a pointer to the created button to allow further configurations.

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 + padding_top + padding_bottom*).

The labels are created with `LV_LABEL_LONG_SCROLL_CIRC` long mode to automatically scroll the long labels circularly.

`lv_list_get_btn_label(list_btn)` and `lv_list_get_btn_img(list_btn)` can be used to get the label and the image of a list button. The text can be set directly with `lv_list_get_btn_text(list_btn)`.

Delete buttons

To delete a list element just use `lv_obj_del(btn)` on the return value of `lv_list_add_btn()`.

To clean the list (remove all buttons) use `lv_list_clean(list)`

Manual navigation

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, LV_ANIM_ON/OFF)`.

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.

Layout

By default the list is vertical. To get a horizontal list use `lv_list_set_layout(list, LV_LAYOUT_ROW_MID)`.

Edge flash

A circle-like effect can be shown when the list reaches the most top or bottom position. `lv_list_set_edge_flash(list, true)` enables this feature.

Scroll propagation

If the list is created on an other scrollable element (like a *Page*) and the list can't be scrolled further the scrolling can be propagated to the parent. This way the scroll will be continued on the parent. It can be enabled with `lv_list_set_scroll_propagation(list, true)`

5.20.4 Events

Only the *Generic events* are sent by the object type.

Learn more about *Events*.

5.20.5 Keys

The following *Keys* are processed by the Lists:

- **LV_KEY_RIGHT/DOWN** Select the next button
- **LV_KEY_LEFT/UP** Select the previous button

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

The Selected buttons are in `LV_BTN_STATE_PR/TG_PR` state.

To manually select a button use `lv_list_set_btn_selected(list, btn)`. When the list is defocused and focused again it will restore the last selected button.

Learn more about *Keys*.

5.20.6 Example

C

Simple List

code

```
#include "../../lv_examples.h"
#include <stdio.h>
#if LV_USE_LIST

static void event_handler(lv_obj_t * obj, lv_event_t event)
{
    if(event == LV_EVENT_CLICKED) {
        printf("Clicked: %s\n", lv_list_get_btn_text(obj));
    }
}

void lv_ex_list_1(void)
{
    /*Create a list*/
    lv_obj_t * list1 = lv_list_create(lv_scr_act(), NULL);
    lv_obj_set_size(list1, 160, 200);
    lv_obj_align(list1, NULL, LV_ALIGN_CENTER, 0, 0);

    /*Add buttons to the list*/
    lv_obj_t * list_btn;

    list_btn = lv_list_add_btn(list1, LV_SYMBOL_FILE, "New");
    lv_obj_set_event_cb(list_btn, event_handler);

    list_btn = lv_list_add_btn(list1, LV_SYMBOL_DIRECTORY, "Open");
    lv_obj_set_event_cb(list_btn, event_handler);

    list_btn = lv_list_add_btn(list1, LV_SYMBOL_CLOSE, "Delete");
    lv_obj_set_event_cb(list_btn, event_handler);

    list_btn = lv_list_add_btn(list1, LV_SYMBOL_EDIT, "Edit");
    lv_obj_set_event_cb(list_btn, event_handler);

    list_btn = lv_list_add_btn(list1, LV_SYMBOL_SAVE, "Save");
    lv_obj_set_event_cb(list_btn, event_handler);

    list_btn = lv_list_add_btn(list1, LV_SYMBOL_BELL, "Notify");
    lv_obj_set_event_cb(list_btn, event_handler);

    list_btn = lv_list_add_btn(list1, LV_SYMBOL_BATTERY_FULL, "Battery");
    lv_obj_set_event_cb(list_btn, event_handler);
}

#endif
```

MicroPython

No examples yet.

5.21 Line meter (lv_lmeter)

5.21.1 Overview

The Line meter object consists of some radial lines which draw a scale. Setting a value for the Line meter will change the color of the scale lines proportionally.

5.21.2 Parts and Styles

The Line meter has only a main part, called `LV_LINEMETER_PART_MAIN`. It uses all the typical background properties to draw a rectangle-like or circle background and the *line* and *scale* properties to draw the scale lines. The active lines (which are related to smaller values than the current value) are colored from *line_color* to *scale_grad_color*. The lines in the end (after the current value) are set to *scale_end_color* color.

5.21.3 Usage

Set value

When setting a new value with `lv_linemeter_set_value(linemeter, new_value)` the proportional part of the scale will be recolored.

Range and Angles

The `lv_linemeter_set_range(linemeter, 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_linemeter_set_scale(linemeter, angle, line_num)`. The default angle is 240 and the default line number is 31.

Angle offset

By default the scale angle is interpreted symmetrically to the y axis. It results in "standing" line meter. With `lv_linemeter_set_angle_offset` an offset can be added to the scale angle. It can be used e.g. to put a quarter line meter into a corner or a half line meter to the right or left side.

Mirror

By default the Line meter's lines are activated clock-wise. It can be changed using `lv_linemeter_set_mirror(linemeter, true/false)`.

5.21.4 Events

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

Learn more about [Events](#).

5.21.5 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.21.6 Example

C

Simple Line meter

code

```
#include "../../lv_examples.h"
#if LV_USE_LINEMETER

void lv_ex_linemeter_1(void)
{
    /*Create a line meter */
    lv_obj_t * lmeter;
    lmeter = lv_linemeter_create(lv_scr_act(), NULL);
    lv_linemeter_set_range(lmeter, 0, 100);           /*Set the range*/
    lv_linemeter_set_value(lmeter, 80);              /*Set the current
↪value*/
    lv_linemeter_set_scale(lmeter, 240, 21);         /*Set the angle and
↪number of lines*/
    lv_obj_set_size(lmeter, 150, 150);
    lv_obj_align(lmeter, NULL, LV_ALIGN_CENTER, 0, 0);
}

#endif
```

MicroPython

No examples yet.

5.22 Message box (lv_msdbox)

5.22.1 Overview

The Message boxes act as pop-ups. They are built from a background *Container*, a *Label* and a *Button matrix* for buttons.

The text will be broken into multiple lines automatically (has `LV_LABEL_LONG_MODE_BREAK`) and the height will be set automatically to involve the text and the buttons (`LV_FIT_TIGHT` fit vertically)-

5.22.2 Parts and Styles

The Message box's main part is called `LV_MSGBOX_PART_MAIN` and it uses all the typical background style properties. Using padding will add space on the sides. *pad_inner* will add space between the text and the buttons. The *label* style properties affect the style of text.

The buttons parts are the same as in case of *Button matrix*:

- `LV_MSGBOX_PART_BTN_BG` the background of the buttons
- `LV_MSGBOX_PART_BTN` the buttons

5.22.3 Usage

Set text

To set the text use the `lv_msgbox_set_text(msgbox, "My text")` function. Not only the pointer of the text will be saved, so the the text can be in a local variable too.

Add buttons

To add buttons use the `lv_msgbox_add_btns(msgbox, btn_str)` function. The button's text needs to be specified like `const char * btn_str[] = {"Apply", "Close", ""}`. For more information visit the *Button matrix* documentation.

The button matrix will be created only when `lv_msgbox_add_btns()` is called for the first time.

Auto-close

With `lv_msgbox_start_auto_close(mbox, delay)` the message box can be closed automatically after `delay` milliseconds with an animation. The `lv_mbox_stop_auto_close(mbox)` function stops a started auto close.

The duration of the close animation can be set by `lv_mbox_set_anim_time(mbox, anim_time)`.

5.22.4 Events

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

- **LV_EVENT_VALUE_CHANGED** sent when the button is clicked. The event data is set to ID of the clicked button.

The Message box has a default event callback which closes itself when a button is clicked.

Learn more about [Events](#).

##Keys

The following [Keys](#) are processed by the Buttons:

- **LV_KEY_RIGHT/DOWN** Select the next button
- **LV_KEY_LEFT/TOP** Select the previous button
- **LV_KEY_ENTER** Clicks the selected button

Learn more about [Keys](#).

5.22.5 Example

C

Simple Message box

code

```
#include "../../lv_examples.h"
#include <stdio.h>
#if LV_USE_MSGBOX

static void event_handler(lv_obj_t * obj, lv_event_t event)
{
    if(event == LV_EVENT_VALUE_CHANGED) {
        printf("Button: %s\n", lv_msgbox_get_active_btn_text(obj));
    }
}

void lv_ex_msgbox_1(void)
{
    static const char * btns[] = {"Apply", "Close", ""};

    lv_obj_t * mbox1 = lv_msgbox_create(lv_scr_act(), NULL);
    lv_msgbox_set_text(mbox1, "A message box with two buttons.");
    lv_msgbox_add_btns(mbox1, btns);
    lv_obj_set_width(mbox1, 200);
    lv_obj_set_event_cb(mbox1, event_handler);
    lv_obj_align(mbox1, NULL, LV_ALIGN_CENTER, 0, 0); /*Align to the corner*/
}

#endif
```

Modal

code

```
#include "../../lv_examples.h"
#if LV_USE_MSGBOX

static void mbox_event_cb(lv_obj_t *obj, lv_event_t evt);
static void btn_event_cb(lv_obj_t *btn, lv_event_t evt);
static void opa_anim(void *bg, lv_anim_value_t v);

static lv_obj_t *mbox, *info;
static lv_style_t style_modal;

static const char welcome_info[] = "Welcome to the modal message box demo!\n"
    "Press the button to display a message box.";

static const char in_msg_info[] = "Notice that you cannot touch "
    "the button again while the message box is open.";

void lv_ex_msgbox_2(void)
{
    lv_style_init(&style_modal);
    lv_style_set_bg_color(&style_modal, LV_STATE_DEFAULT, LV_COLOR_BLACK);

    /* Create a button, then set its position and event callback */
    lv_obj_t *btn = lv_btn_create(lv_scr_act(), NULL);
    lv_obj_set_size(btn, 200, 60);
    lv_obj_set_event_cb(btn, btn_event_cb);
    lv_obj_align(btn, NULL, LV_ALIGN_IN_TOP_LEFT, 20, 20);

    /* Create a label on the button */
    lv_obj_t *label = lv_label_create(btn, NULL);
    lv_label_set_text(label, "Display a message box!");

    /* Create an informative label on the screen */
    info = lv_label_create(lv_scr_act(), NULL);
    lv_label_set_text(info, welcome_info);
    lv_label_set_long_mode(info, LV_LABEL_LONG_BREAK); /* Make sure text will wrap */
    lv_obj_set_width(info, LV_HOR_RES - 10);
    lv_obj_align(info, NULL, LV_ALIGN_IN_BOTTOM_LEFT, 5, -5);
}

static void mbox_event_cb(lv_obj_t *obj, lv_event_t evt)
{
    if(evt == LV_EVENT_DELETE && obj == mbox) {
        /* Delete the parent modal background */
        lv_obj_del_async(lv_obj_get_parent(mbox));
        mbox = NULL; /* happens before object is actually deleted! */
        lv_label_set_text(info, welcome_info);
    } else if(evt == LV_EVENT_VALUE_CHANGED) {
        /* A button was clicked */
        lv_msgbox_start_auto_close(mbox, 0);
    }
}
}
```

(continues on next page)

(continued from previous page)

```

static void btn_event_cb(lv_obj_t *btn, lv_event_t evt)
{
    if(evt == LV_EVENT_CLICKED) {
        /* Create a full-screen background */

        /* Create a base object for the modal background */
        lv_obj_t *obj = lv_obj_create(lv_scr_act(), NULL);
        lv_obj_reset_style_list(obj, LV_OBJ_PART_MAIN);
        lv_obj_add_style(obj, LV_OBJ_PART_MAIN, &style_modal);
        lv_obj_set_pos(obj, 0, 0);
        lv_obj_set_size(obj, LV_HOR_RES, LV_VER_RES);

        static const char * btns2[] = {"Ok", "Cancel", ""};

        /* Create the message box as a child of the modal background */
        mbox = lv_msgbox_create(obj, NULL);
        lv_msgbox_add_btns(mbox, btns2);
        lv_msgbox_set_text(mbox, "Hello world!");
        lv_obj_align(mbox, NULL, LV_ALIGN_CENTER, 0, 0);
        lv_obj_set_event_cb(mbox, mbox_event_cb);

        /* Fade the message box in with an animation */
        lv_anim_t a;
        lv_anim_init(&a);
        lv_anim_set_var(&a, obj);
        lv_anim_set_time(&a, 500);
        lv_anim_set_values(&a, LV_OPA_TRANSP, LV_OPA_50);
        lv_anim_set_exec_cb(&a, (lv_anim_exec_xcb_t)opa_anim);
        lv_anim_start(&a);

        lv_label_set_text(info, in_msg_info);
        lv_obj_align(info, NULL, LV_ALIGN_IN_BOTTOM_LEFT, 5, -5);
    }
}

static void opa_anim(void * bg, lv_anim_value_t v)
{
    lv_obj_set_style_local_bg_opa(bg, LV_OBJ_PART_MAIN, LV_STATE_DEFAULT, v);
}

#endif

```

MicroPython

No examples yet.

5.23 Object mask (lv_objmask)

5.23.1 Overview

The *Object mask* is capable of add some mask to drawings when its children is drawn.

5.23.2 Parts and Styles

The Object mask has only a main part called `LV_OBJMASK_PART_BG` and it uses the typical background style properties.

5.23.3 Usage

Adding mask

Before adding a mask to the *Object mask* the mask should be initialized:

```
lv_draw_mask_<type>_param_t mask_param;
lv_draw_mask_<type>_init(&mask_param, ...);
lv_objmask_mask_t * mask_p = lv_objmask_add_mask(objmask, &mask_param);
```

Lvgl supports the following mask types:

- **line** clip the pixels on the top/bottom left/right of a line. Can be initialized from two points or a point and an angle:
- **angle** keep the pixels only between a given start and end angle
- **radius** keep the pixel only inside a rectangle which can have radius (can for a circle too). Can be inverted to keep the pixel outside of the rectangle.
- **fade** fade vertically (change the pixels opacity according to their y position)
- **map** use an alpha mask (a byte array) to describe the pixels opacity.

The coordinates in the mask are relative to the Object. That is if the object moves the masks move with it. For the details of the mask *init* function see the *API* documentation below.

Update mask

AN existing mask can be updated with `lv_objmask_update_mask(objmask, mask_p, new_param)`, where `mask_p` is return value of `lv_objmask_add_mask`.

Remove mask

A mask can be removed with `lv_objmask_remove_mask(objmask, mask_p)`

5.23.4 Events

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

Learn more about [Events](#).

5.23.5 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.23.6 Example

C

Several object masks

code

```
#include "../../lv_examples.h"
#if LV_USE_OBJMASK

void lv_ex_objmask_1(void)
{
    /*Set a very visible color for the screen to clearly see what happens*/
    lv_obj_set_style_local_bg_color(lv_scr_act(), LV_OBJ_PART_MAIN, LV_STATE_DEFAULT,
    ↪ lv_color_hex3(0xf33));

    lv_obj_t * om = lv_objmask_create(lv_scr_act(), NULL);
    lv_obj_set_size(om, 200, 200);
    lv_obj_align(om, NULL, LV_ALIGN_CENTER, 0, 0);
    lv_obj_t * label = lv_label_create(om, NULL);
    lv_label_set_long_mode(label, LV_LABEL_LONG_BREAK);
    lv_label_set_align(label, LV_LABEL_ALIGN_CENTER);
    lv_obj_set_width(label, 180);
    lv_label_set_text(label, "This label will be masked out. See how it works.");
    lv_obj_align(label, NULL, LV_ALIGN_IN_TOP_MID, 0, 20);

    lv_obj_t * cont = lv_cont_create(om, NULL);
    lv_obj_set_size(cont, 180, 100);
    lv_obj_set_drag(cont, true);
    lv_obj_align(cont, NULL, LV_ALIGN_IN_BOTTOM_MID, 0, -10);

    lv_obj_t * btn = lv_btn_create(cont, NULL);
    lv_obj_align(btn, NULL, LV_ALIGN_CENTER, 0, 0);
    lv_obj_set_style_local_value_str(btn, LV_BTN_PART_MAIN, LV_STATE_DEFAULT, "Button
    ↪");
    uint32_t t;
```

(continues on next page)

(continued from previous page)

```

lv_refr_now(NULL);
t = lv_tick_get();
while(lv_tick_elaps(t) < 1000);

lv_area_t a;
lv_draw_mask_radius_param_t r1;

a.x1 = 10;
a.y1 = 10;
a.x2 = 190;
a.y2 = 190;
lv_draw_mask_radius_init(&r1, &a, LV_RADIUS_CIRCLE, false);
lv_objmask_add_mask(om, &r1);

lv_refr_now(NULL);
t = lv_tick_get();
while(lv_tick_elaps(t) < 1000);

a.x1 = 100;
a.y1 = 100;
a.x2 = 150;
a.y2 = 150;
lv_draw_mask_radius_init(&r1, &a, LV_RADIUS_CIRCLE, true);
lv_objmask_add_mask(om, &r1);

lv_refr_now(NULL);
t = lv_tick_get();
while(lv_tick_elaps(t) < 1000);

lv_draw_mask_line_param_t l1;
lv_draw_mask_line_points_init(&l1, 0, 0, 100, 200, LV_DRAW_MASK_LINE_SIDE_TOP);
lv_objmask_add_mask(om, &l1);

lv_refr_now(NULL);
t = lv_tick_get();
while(lv_tick_elaps(t) < 1000);

lv_draw_mask_fade_param_t f1;
a.x1 = 100;
a.y1 = 0;
a.x2 = 200;
a.y2 = 200;
lv_draw_mask_fade_init(&f1, &a, LV_OPA_TRANSP, 0, LV_OPA_COVER, 150);
lv_objmask_add_mask(om, &f1);
}

#endif

```

Text mask

code

```

#include "../../lv_examples.h"
#if LV_USE_OBJMASK

#define MASK_WIDTH 100
#define MASK_HEIGHT 50

void lv_ex_objmask_2(void)
{
    /* Create the mask of a text by drawing it to a canvas*/
    static lv_opa_t mask_map[MASK_WIDTH * MASK_HEIGHT];

    /*Create a "8 bit alpha" canvas and clear it*/
    lv_obj_t * canvas = lv_canvas_create(lv_scr_act(), NULL);
    lv_canvas_set_buffer(canvas, mask_map, MASK_WIDTH, MASK_HEIGHT, LV_IMG_CF_ALPHA_
↪8BIT);
    lv_canvas_fill_bg(canvas, LV_COLOR_BLACK, LV_OPA_TRANSP);

    /*Draw a label to the canvas. The result "image" will be used as mask*/
    lv_draw_label_dsc_t label_dsc;
    lv_draw_label_dsc_init(&label_dsc);
    label_dsc.color = LV_COLOR_WHITE;
    lv_canvas_draw_text(canvas, 5, 5, MASK_WIDTH, &label_dsc, "Text with gradient",
↪LV_LABEL_ALIGN_CENTER);

    /*The mask is reads the canvas is not required anymore*/
    lv_obj_del(canvas);

    /*Create an object mask which will use the created mask*/
    lv_obj_t * om = lv_objmask_create(lv_scr_act(), NULL);
    lv_obj_set_size(om, MASK_WIDTH, MASK_HEIGHT);
    lv_obj_align(om, NULL, LV_ALIGN_CENTER, 0, 0);

    /*Add the created mask map to the object mask*/
    lv_draw_mask_map_param_t m;
    lv_area_t a;
    a.x1 = 0;
    a.y1 = 0;
    a.x2 = MASK_WIDTH - 1;
    a.y2 = MASK_HEIGHT - 1;
    lv_draw_mask_map_init(&m, &a, mask_map);
    lv_objmask_add_mask(om, &m);

    /*Create a style with gradient*/
    static lv_style_t style_bg;
    lv_style_init(&style_bg);
    lv_style_set_bg_opa(&style_bg, LV_STATE_DEFAULT, LV_OPA_COVER);
    lv_style_set_bg_color(&style_bg, LV_STATE_DEFAULT, LV_COLOR_RED);
    lv_style_set_bg_grad_color(&style_bg, LV_STATE_DEFAULT, LV_COLOR_BLUE);
    lv_style_set_bg_grad_dir(&style_bg, LV_STATE_DEFAULT, LV_GRAD_DIR_HOR);

    /* Create and object with the gradient style on the object mask.
     * The text will be masked from the gradient*/

```

(continues on next page)

(continued from previous page)

```

lv_obj_t * bg = lv_obj_create(om, NULL);
lv_obj_reset_style_list(bg, LV_OBJ_PART_MAIN);
lv_obj_add_style(bg, LV_OBJ_PART_MAIN, &style_bg);
lv_obj_set_size(bg, MASK_WIDTH, MASK_HEIGHT);
}

#endif

```

MicroPython

No examples yet.

5.24 Page (lv_page)

5.24.1 Overview

The Page consist of two *Containers* on each other:

- a **background**
- a top which is **scrollable**.

5.24.2 Parts and Styles

The Page's main part is called **LV_PAGE_PART_BG** which is the background of the Page. It uses all the typical background style properties. Using padding will add space on the sides.

The scrollable object can be referenced via the **LV_PAGE_PART_SCRL** part. It also uses all the typical background style properties and padding to add space on the sides.

LV_LIST_PART_SCROLLBAR is a virtual part of the background to draw the scroll bars. Uses all the typical background style properties, *size* to set the width of the scroll bars, and *pad_right* and *pad_bottom* to set the spacing.

LV_LIST_PART_EDGE_FLASH is also a virtual part of the background to draw a semicircle on the sides when the list can not be scrolled in that direction further. Uses all the typical background properties.

5.24.3 Usage

The background object can be referenced as the page itself like. E.g. to set the page's width: `lv_obj_set_width(page, 100)`.

If a child is created on the page it will be automatically moved to the scrollable container. If the scrollable container becomes larger then the background it can be scrolled by dragging (like the lists on smartphones).

By default, the scrollable's has **LV_FIT_MAX** fit in all directions. It means the scrollable size will be the same as the background's size (minus the padding) while the children are in the background. But when an object is positioned out of the background the scrollable size will be increased to involve it.

Scrollbars

Scrollbars can be shown according to four policies:

- `LV_SCROLLBAR_MODE_OFF` Never show scroll bars
- `LV_SCROLLBAR_MODE_ON` Always show scroll bars
- `LV_SCROLLBAR_MODE_DRAG` Show scroll bars when the page is being dragged
- `LV_SCROLLBAR_MODE_AUTO` Show scroll bars when the scrollable container is large enough to be scrolled
- `LV_SCROLLBAR_MODE_HIDE` Hide the scroll bar temporally
- `LV_SCROLLBAR_MODE_UNHIDE` Unhide the previously hidden scrollbar. Recover the original mode too

The scroll bar show policy can be changed by: `lv_page_set_scrollbar_mode(page, SB_MODE)`. The default value is `LV_SCROLLBAR_MODE_AUTO`.

Glue object

A children can be "glued" to the page. In this case, if the page can be scrolled by dragging that object. It can be enabled by the `lv_page_glue_obj(child, true)`.

Focus object

An object on a page can be focused with `lv_page_focus(page, child, LV_ANIM_ON/OFF)`. It will move the scrollable container to show a child. The time of the animation can be set by `lv_page_set_anim_time(page, anim_time)` in milliseconds. `child` doesn't have to be a direct child of the page. This is it works if the scrollable object is the grandparent of the object too.

Manual navigation

You can move the scrollable object manually using `lv_page_scroll_hor(page, dist)` and `lv_page_scroll_ver(page, dist)`

Edge flash

A circle-like effect can be shown if the list reached the most top/bottom/left/right position. `lv_page_set_edge_flash(list, en)` enables this feature.

Scroll propagation

If the list is created on an other scrollable element (like an other page) and the Page can't be scrolled further the scrolling can be propagated to the parent to continue the scrolling on the parent. It can be enabled with `lv_page_set_scroll_propagation(list, true)`

5.24.4 Clean the page

All the object created on the page can be clean with `lv_page_clean(page)`. Note that `lv_obj_clean(page)` doesn't work here because it would delete the scrollable object too.

Scrollable API

There are functions to directly set/get the scrollable's attributes:

- `lv_page_get_scr1()`
- `lv_page_set_scr1_fit/fint2/fit4()`
- `lv_page_set_scr1_width()`
- `lv_page_set_scr1_height()`
- `lv_page_set_scr1_fit_width()`
- `lv_page_set_scr1_fit_height()`
- `lv_page_set_scr1_layout()`

5.24.5 Events

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

The scrollable object has a default event callback which propagates the following events to the background object: `LV_EVENT_PRESSED`, `LV_EVENT_PRESSING`, `LV_EVENT_PRESS_LOST`, `LV_EVENT_RELEASED`, `LV_EVENT_SHORT_CLICKED`, `LV_EVENT_CLICKED`, `LV_EVENT_LONG_PRESSED`, `LV_EVENT_LONG_PRESSED_REPEAT`

Learn more about [Events](#).

##Keys

The following *Keys* are processed by the Page:

- `LV_KEY_RIGHT/LEFT/UP/DOWN` Scroll the page

Learn more about [Keys](#).

5.24.6 Example

C

Page with scrollbar

code

```
#include "../../lv_examples.h"
#if LV_USE_PAGE

void lv_ex_page_1(void)
{
    /*Create a page*/
    lv_obj_t * page = lv_page_create(lv_scr_act(), NULL);
```

(continues on next page)

(continued from previous page)

```

lv_obj_set_size(page, 150, 200);
lv_obj_align(page, NULL, LV_ALIGN_CENTER, 0, 0);

/*Create a label on the page*/
lv_obj_t * label = lv_label_create(page, NULL);
lv_label_set_long_mode(label, LV_LABEL_LONG_BREAK);           /*Automatically
↪break long lines*/
lv_obj_set_width(label, lv_page_get_width_fit(page));         /*Set the label
↪width to max value to not show hor. scroll bars*/
lv_label_set_text(label, "Lorem ipsum dolor sit amet, consectetur adipiscing elit,
↪\n"
                                "sed do eiusmod tempor incididunt ut labore et dolore
↪magna aliqua.\n"
                                "Ut enim ad minim veniam, quis nostrud exercitation
↪ullamco\n"
                                "laboris nisi ut aliquip ex ea commodo consequat. Duis
↪aute irure\n"
                                "dolor in reprehenderit in voluptate velit esse cillum
↪dolore\n"
                                "eu fugiat nulla pariat.\n"
                                "Excepteur sint occaecat cupidatat non proident, sunt in
↪culpa\n"
                                "qui officia deserunt mollit anim id est laborum.");
}

#endif

```

MicroPython

No examples yet.

5.25 Roller (lv_roller)

5.25.1 Overview

Roller allows you to simply select one option from more with scrolling.

5.25.2 Parts and Styles

The Roller's main part is called **LV_ROLLER_PART_BG**. It's a rectangle and uses all the typical background properties. The style of the Roller's label is inherited from the *text* style properties of the background. To adjust the space between the options use the *text_line_space* style property. The *padding* style properties set the space on the sides.

The selected option in the middle can be referenced with **LV_ROLLER_PART_SELECTED** virtual part. Besides the typical background properties it uses the *text* properties to change the appearance of the text in the selected area.

5.25.3 Usage

Set options

The options are passed to the Roller as a string with `lv_roller_set_options(roller, options, LV_ROLLER_MODE_NORMAL/INFINITE)`. The options should be separated by `\n`. For example: "First\nSecond\nThird".

`LV_ROLLER_MODE_INIFINITE` make the roller circular.

You can select an option manually with `lv_roller_set_selected(roller, id, LV_ANIM_ON/OFF)`, where *id* is the index of an option.

Get selected option

To get the currently selected option use `lv_roller_get_selected(roller)` it will return the *index* of the selected option.

`lv_roller_get_selected_str(roller, buf, buf_size)` copy the name of the selected option to *buf*.

Align the options

To align the label horizontally use `lv_roller_set_align(roller, LV_LABEL_ALIGN_LEFT/CENTER/RIGHT)`.

Visible rows

The number of visible rows can be adjusted with `lv_roller_set_visible_row_count(roller, num)`

Animation time

When the Roller is scrolled and doesn't stop exactly on an option it will scroll to the nearest valid option automatically. The time of this scroll animation can be changed by `lv_roller_set_anim_time(roller, anim_time)`. Zero animation time means no animation.

5.25.4 Events

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

- `LV_EVENT_VALUE_CHANGED` sent when a new option is selected

Learn more about [Events](#).

5.25.5 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)

5.25.6 Example

C

Simple Roller

code

```
#include "../../lv_examples.h"
#include <stdio.h>
#if LV_USE_ROLLER

static void event_handler(lv_obj_t * obj, lv_event_t event)
{
    if(event == LV_EVENT_VALUE_CHANGED) {
        char buf[32];
        lv_roller_get_selected_str(obj, buf, sizeof(buf));
        printf("Selected month: %s\n", buf);
    }
}

void lv_ex_roller_1(void)
{
    lv_obj_t *roller1 = lv_roller_create(lv_scr_act(), NULL);
    lv_roller_set_options(roller1,
        "January\n"
        "February\n"
        "March\n"
        "April\n"
        "May\n"
        "June\n"
        "July\n"
        "August\n"
        "September\n"
        "October\n"
        "November\n"
        "December",
        LV_ROLLER_MODE_INFINITE);

    lv_roller_set_visible_row_count(roller1, 4);
    lv_obj_align(roller1, NULL, LV_ALIGN_CENTER, 0, 0);
    lv_obj_set_event_cb(roller1, event_handler);
}

#endif
```

MicroPython

No examples yet.

5.26 Slider (lv_slider)

5.26.1 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.

5.26.2 Parts and Styles

The Slider's main part is called `LV_SLIDER_PART_BG` and it uses the typical background style properties.

`LV_SLIDER_PART_INDIC` is a virtual part which also uses all the typical background properties. By default, the indicator maximal size is the same as the background's size but setting positive padding values in `LV_SLIDER_PART_BG` will make the indicator smaller. (negative values will make it larger) If the *value* style property is used on the indicator the alignment will be calculated based on the current size of the indicator. For example a center aligned value is always shown in the middle of the indicator regardless it's current size.

`LV_SLIDER_PART_KNOB` is a virtual part using all the typical background properties to describe the knob(s). Similarly to the *indicator* the *value* text is also aligned to the current position and size of the knob. By default the knob is square (with a radius) with side length equal to the smaller side of the slider. The knob can be made larger with the *padding* values. Padding values can be asymmetric too.

5.26.3 Usage

Value and range

To set an initial value use `lv_slider_set_value(slider, new_value, LV_ANIM_ON/OFF)`. `lv_slider_set_anim_time(slider, anim_time)` sets the animation time in milliseconds.

To specify the range (min, max values) the `lv_slider_set_range(slider, min , max)` can be used.

Symmetrical and Range

Besides the normal type the Slider can be configured in two additional types:

- `LV_SLIDER_TYPE_NORMAL` normal type
- `LV_SLIDER_TYPE_SYMMETRICAL` draw the indicator symmetrical to zero (drawn from zero, left to right)
- `LV_SLIDER_TYPE_RANGE` allow the use of an additional knob for the left (start) value. (Can be used with `lv_slider_set/get_left_value()`)

The type can be changed with `lv_slider_set_type(slider, LV_SLIDER_TYPE_...)`

Knob-only mode

Normally, the slider can be adjusted either by dragging the knob, or clicking on the slider bar. In the latter case the knob moves to the point clicked and slider value changes accordingly. In some cases it is desirable to set the slider to react on dragging the knob only.

This feature is enabled by calling `lv_obj_set_adv_hittest(slider, true);`.

5.26.4 Events

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

- **LV_EVENT_VALUE_CHANGED** Sent while the slider is being dragged or changed with keys. The event is sent continuously while the slider is dragged and only when it is released. Use `lv_slider_is_dragged` to decide whether is slider is being dragged or just released.

5.26.5 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

Learn more about [Keys](#).

5.26.6 Example

C

Slider with custom style

code

```
#include "../../lv_examples.h"
#include <stdio.h>
#if LV_USE_SLIDER

static void event_handler(lv_obj_t * obj, lv_event_t event)
{
    if(event == LV_EVENT_VALUE_CHANGED) {
        printf("Value: %d\n", lv_slider_get_value(obj));
    }
}

void lv_ex_slider_1(void)
{
    /*Create a slider*/
    lv_obj_t * slider = lv_slider_create(lv_scr_act(), NULL);
    lv_obj_align(slider, NULL, LV_ALIGN_CENTER, 0, 0);
    lv_obj_set_event_cb(slider, event_handler);
}

#endif
```

Set value with slider

code

```

#include "../../lv_examples.h"
#include <stdio.h>
#if LV_USE_SLIDER

static void slider_event_cb(lv_obj_t * slider, lv_event_t event);
static lv_obj_t * slider_label;

void lv_ex_slider_2(void)
{
    /* Create a slider in the center of the display */
    lv_obj_t * slider = lv_slider_create(lv_scr_act(), NULL);
    lv_obj_set_width(slider, LV_DPI * 2);
    lv_obj_align(slider, NULL, LV_ALIGN_CENTER, 0, 0);
    lv_obj_set_event_cb(slider, slider_event_cb);
    lv_slider_set_range(slider, 0, 100);

    /* Create a label below the slider */
    slider_label = lv_label_create(lv_scr_act(), NULL);
    lv_label_set_text(slider_label, "0");
    lv_obj_set_auto_realign(slider_label, true);
    lv_obj_align(slider_label, slider, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);

    /* Create an informative label */
    lv_obj_t * info = lv_label_create(lv_scr_act(), NULL);
    lv_label_set_text(info, "Welcome to the slider+label demo!\n"
                           "Move the slider and see that the label\n"
                           "updates to match it.");
    lv_obj_align(info, NULL, LV_ALIGN_IN_TOP_LEFT, 10, 10);
}

static void slider_event_cb(lv_obj_t * slider, lv_event_t event)
{
    if(event == LV_EVENT_VALUE_CHANGED) {
        static char buf[4]; /* max 3 bytes for number plus 1 null terminating byte */
        snprintf(buf, 4, "%u", lv_slider_get_value(slider));
        lv_label_set_text(slider_label, buf);
    }
}

#endif

```

MicroPython

No examples yet.

5.27 Spinbox (lv_spinbox)

5.27.1 Overview

The Spinbox contains a number as text which can be increased or decreased by *Keys* or API functions. Under the hood the Spinbox is a modified *Text area*.

5.27.2 Parts and Styles

The Spinbox's main part is called `LV_SPINBOX_PART_BG` which is a rectangle-like background using all the typical background style properties. It also describes the style of the label with its *text* style properties.

`LV_SPINBOX_PART_CURSOR` is a virtual part describing the cursor. Read the *Text area* documentation for a detailed description.

Set format

`lv_spinbox_set_digit_format(spinbox, digit_count, separator_position)` set the format of the number. `digit_count` sets the number of digits. Leading zeros are added to fill the space on the left. `separator_position` sets the number of digit before the decimal point. `0` means no decimal point.

`lv_spinbox_set_padding_left(spinbox, cnt)` add `cnt` "space" characters between the sign and the most left digit.

Value and ranges

`lv_spinbox_set_range(spinbox, min, max)` sets the range of the Spinbox.

`lv_spinbox_set_value(spinbox, num)` sets the Spinbox's value manually.

`lv_spinbox_increment(spinbox)` and `lv_spinbox_decrement(spinbox)` increments/decrements the value of the Spinbox.

`lv_spinbox_set_step(spinbox, step)` sets the amount to increment/decrement.

5.27.3 Events

Besides the *Generic events* the following *Special events* are sent by the Drop down lists:

- **LV_EVENT_VALUE_CHANGED** sent when the value has changed. (the value is set as event data as `int32_t`)
- **LV_EVENT_INSERT** sent by the ancestor Text area but shouldn't be used.

Learn more about *Events*.

5.27.4 Keys

The following *Keys* are processed by the Buttons:

- **LV_KEY_LEFT/RIGHT** With *Keypad* move the cursor left/right. With *Encoder* decrement/increment the selected digit.
- **LV_KEY_ENTER** Apply the selected option (Send **LV_EVENT_VALUE_CHANGED** event and close the Drop down list)
- **LV_KEY_ENTER** With *Encoder* got the net digit. Jump to the first after the last.

5.27.5 Example

C

Simple Spinbox

code

```
#include "../../lv_examples.h"
#include <stdio.h>
#if LV_USE_SPINBOX

static lv_obj_t * spinbox;

static void lv_spinbox_increment_event_cb(lv_obj_t * btn, lv_event_t e)
{
    if(e == LV_EVENT_SHORT_CLICKED || e == LV_EVENT_LONG_PRESSED_REPEAT) {
        lv_spinbox_increment(spinbox);
    }
}

static void lv_spinbox_decrement_event_cb(lv_obj_t * btn, lv_event_t e)
{
    if(e == LV_EVENT_SHORT_CLICKED || e == LV_EVENT_LONG_PRESSED_REPEAT) {
        lv_spinbox_decrement(spinbox);
    }
}

void lv_ex_spinbox_1(void)
{
    spinbox = lv_spinbox_create(lv_scr_act(), NULL);
    lv_spinbox_set_range(spinbox, -1000, 90000);
    lv_spinbox_set_digit_format(spinbox, 5, 2);
    lv_spinbox_step_prev(spinbox);
    lv_obj_set_width(spinbox, 100);
    lv_obj_align(spinbox, NULL, LV_ALIGN_CENTER, 0, 0);

    lv_coord_t h = lv_obj_get_height(spinbox);
    lv_obj_t * btn = lv_btn_create(lv_scr_act(), NULL);
    lv_obj_set_size(btn, h, h);
    lv_obj_align(btn, spinbox, LV_ALIGN_OUT_RIGHT_MID, 5, 0);
    lv_theme_apply(btn, LV_THEME_SPINBOX_BTN);
}
```

(continues on next page)

(continued from previous page)

```

    lv_obj_set_style_local_value_str(btn, LV_BTN_PART_MAIN, LV_STATE_DEFAULT, LV_
↪SYMBOL_PLUS);
    lv_obj_set_event_cb(btn, lv_spinbox_increment_event_cb);

    btn = lv_btn_create(lv_scr_act(), btn);
    lv_obj_align(btn, spinbox, LV_ALIGN_OUT_LEFT_MID, -5, 0);
    lv_obj_set_event_cb(btn, lv_spinbox_decrement_event_cb);
    lv_obj_set_style_local_value_str(btn, LV_BTN_PART_MAIN, LV_STATE_DEFAULT, LV_
↪SYMBOL_MINUS);
}

#endif

```

MicroPython

No examples yet.

5.28 Spinner (lv_spinner)

5.28.1 Overview

The Spinner object is a spinning arc over a border.

5.28.2 Parts and Styles

The Spinner uses the the following parts:

- LV_SPINNER_PART_BG: main part
- LV_SPINNER_PART_INDIC: the spinning arc (virtual part)

The parts and style works the same as in case of [Arc](#). Read its documentation for a details description.

5.28.3 Usage

Arc length

The length of the arc can be adjusted by `lv_spinner_set_arc_length(spinner, deg)`.

Spinning speed

The speed of the spinning can be adjusted by `lv_spinner_set_spin_time(preload, time_ms)`.

Spin types

You can choose from more spin types:

- **LV_SPINNER_TYPE_SPINNING_ARC** spin the arc, slow down on the top
- **LV_SPINNER_TYPE_FILLSPIN_ARC** spin the arc, slow down on the top but also stretch the arc
- **LV_SPINNER_TYPE_CONSTANT_ARC** spin the arc at a constant speed

To apply one if them use `lv_spinner_set_type(preload, LV_SPINNER_TYPE_...)`

Spin direction

The direction of spinning can be changed with `lv_spinner_set_dir(preload, LV_SPINNER_DIR_FORWARD/BACKWARD)`.

5.28.4 Events

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

5.28.5 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.28.6 Example

C

Simple spinner

code

```
#include "../../lv_examples.h"
#if LV_USE_SPINNER

void lv_ex_spinner_1(void)
{
    /*Create a Preloader object*/
    lv_obj_t * preload = lv_spinner_create(lv_scr_act(), NULL);
    lv_obj_set_size(preload, 100, 100);
    lv_obj_align(preload, NULL, LV_ALIGN_CENTER, 0, 0);
}

#endif
```

MicroPython

MicroPython

No examples yet.

5.29 Switch (lv_switch)

5.29.1 Overview

The Switch can be used to turn on/off something. It looks like a little slider.

5.29.2 Parts and Styles

The Switch uses the the following parts:

- `LV_SWITCH_PART_BG`: main part
- `LV_SWITCH_PART_INDIC`: the indicator (virtual part)
- `LV_SWITCH_PART_KNOB`: the knob (virtual part)

The parts and style works the same as in case of *Slider*. Read its documentation for a details description.

##Usage

Change state

The state of the Switch can be changed by clicking on it or by `lv_switch_on(switch, LV_ANIM_ON/OFF)`, `lv_switch_off(switch, LV_ANIM_ON/OFF)` or `lv_switch_toggle(switch, LV_ANOM_ON/OFF)` functions

Animation time

The time of animations, when the switch changes state, can be adjusted with `lv_switch_set_anim_time(switch, anim_time)`.

5.29.3 Events

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

- `LV_EVENT_VALUE_CHANGED` Sent when the switch changes state.

5.29.4 Keys

- **LV_KEY_UP**, **LV_KEY_RIGHT** Turn on the slider
- **LV_KEY_DOWN**, **LV_KEY_LEFT** Turn off the slider

Learn more about *Keys*.

5.29.5 Example

C

Simple Switch

code

```
#include "../../lv_examples.h"
#include <stdio.h>
#if LV_USE_SWITCH

static void event_handler(lv_obj_t * obj, lv_event_t event)
{
    if(event == LV_EVENT_VALUE_CHANGED) {
        printf("State: %s\n", lv_switch_get_state(obj) ? "On" : "Off");
    }
}

void lv_ex_switch_1(void)
{
    /*Create a switch and apply the styles*/
    lv_obj_t *sw1 = lv_switch_create(lv_scr_act(), NULL);
    lv_obj_align(sw1, NULL, LV_ALIGN_CENTER, 0, -50);
    lv_obj_set_event_cb(sw1, event_handler);

    /*Copy the first switch and turn it ON*/
    lv_obj_t *sw2 = lv_switch_create(lv_scr_act(), sw1);
    lv_switch_on(sw2, LV_ANIM_ON);
    lv_obj_align(sw2, NULL, LV_ALIGN_CENTER, 0, 50);
}

#endif
```

MicroPython

No examples yet.

5.30 Table (lv_table)

5.30.1 Overview

Tables, as usual, are built from rows, columns, and cells containing texts.

The Table object is very light weighted because only the texts are stored. No real objects are created for cells but they are just drawn on the fly.

5.30.2 Parts and Styles

The main part of the Table is called `LV_TABLE_PART_BG`. It's a rectangle like background and uses all the typical background style properties.

For the cells there are 4 virtual parts. Every cell has type (1, 2, 3 or 4) which tells which part's styles to apply on them. The cell parts are:

- `LV_TABLE_PART_CELL1`
- `LV_TABLE_PART_CELL2`
- `LV_TABLE_PART_CELL3`
- `LV_TABLE_PART_CELL4`

The cells also use all the typical background style properties. If there is a line break (`\n`) in a cell's content then a horizontal division line will drawn after the line break using the *line* style properties.

The style of texts in the cells are inherited from the cell parts or the background part.

5.30.3 Usage

Rows and Columns

To set number of rows and columns use `lv_table_set_row_cnt(table, row_cnt)` and `lv_table_set_col_cnt(table, col_cnt)`

Width and Height

The width of the columns can be set with `lv_table_set_col_width(table, col_id, width)`. The overall width of the Table object will be set to the sum of columns widths.

The height is calculated automatically from the cell styles (font, padding etc) and the number of rows.

Set cell value

The cells can store only texts so numbers needs to be converted to text before displaying them in a table.

`lv_table_set_cell_value(table, row, col, "Content")`. The text is saved by the table so it can be even a local variable.

Line break can be used in the text like `"Value\n60.3"`.

Align

The text alignment in cells can be adjusted individually with `lv_table_set_cell_align(table, row, col, LV_LABEL_ALIGN_LEFT/CENTER/RIGHT)`.

Cell type

You can use 4 different cell types. Each has its own style.

Cell types can be used to add different style for example to:

- table header
- first column
- highlight a cell
- etc

The type can be selected with `lv_table_set_cell_type(table, row, col, type)` type can be 1, 2, 3 or 4.

Merge cells

Cells can be merged horizontally with `lv_table_set_cell_merge_right(table, col, row, true)`. To merge more adjacent cells apply this function for each cell.

Crop text

By default, the texts are word-wrapped to fit into the width of the cell and the height of the cell is set automatically. To disable this and keep the text as it is enable `lv_table_set_cell_crop(table, row, col, true)`.

Scroll

To make the Table scrollable place it on a [Page](#)

5.30.4 Events

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

Learn more about [Events](#).

5.30.5 Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

5.30.6 Example

C

Simple table

code

```
#include "../../lv_examples.h"
#if LV_USE_TABLE

void lv_ex_table_1(void)
{
    lv_obj_t * table = lv_table_create(lv_scr_act(), NULL);
    lv_table_set_col_cnt(table, 2);
    lv_table_set_row_cnt(table, 4);
    lv_obj_align(table, NULL, LV_ALIGN_CENTER, 0, 0);

    /*Make the cells of the first row center aligned */
    lv_table_set_cell_align(table, 0, 0, LV_LABEL_ALIGN_CENTER);
    lv_table_set_cell_align(table, 0, 1, LV_LABEL_ALIGN_CENTER);

    /*Align the price values to the right in the 2nd column*/
    lv_table_set_cell_align(table, 1, 1, LV_LABEL_ALIGN_RIGHT);
    lv_table_set_cell_align(table, 2, 1, LV_LABEL_ALIGN_RIGHT);
    lv_table_set_cell_align(table, 3, 1, LV_LABEL_ALIGN_RIGHT);

    lv_table_set_cell_type(table, 0, 0, 2);
    lv_table_set_cell_type(table, 0, 1, 2);

    /*Fill the first column*/
    lv_table_set_cell_value(table, 0, 0, "Name");
    lv_table_set_cell_value(table, 1, 0, "Apple");
    lv_table_set_cell_value(table, 2, 0, "Banana");
    lv_table_set_cell_value(table, 3, 0, "Citron");

    /*Fill the second column*/
    lv_table_set_cell_value(table, 0, 1, "Price");
    lv_table_set_cell_value(table, 1, 1, "$7");
    lv_table_set_cell_value(table, 2, 1, "$4");
    lv_table_set_cell_value(table, 3, 1, "$6");
}

#endif
```

MicroPython

No examples yet.

5.31 Tabview (lv_tabview)

5.31.1 Overview

The Tab view object can be used to organize content in tabs.

5.31.2 Parts and Styles

The Tab view object has several parts. The main is `LV_TABVIEW_PART_BG`. It a rectangle-like container which holds the other parts of the Tab view.

On the background 2 important real parts are created:

- `LV_TABVIEW_PART_BG_SCRL`: it's the scrollable part of *Page*. It holds the content of the tabs next to each other. The background of the Page is always transparent and can't be accessed externally.
- `LV_TABVIEW_PART_TAB_BG`: The tab buttons which is a *Button matrix*. Clicking on a button will scroll `LV_TABVIEW_PART_BG_SCRL` to the related tab's content. The tab buttons can be accessed via `LV_TABVIEW_PART_TAB_BTN`. When tabs are selected, the buttons are in the checked state, and can be styled using `LV_STATE_CHECKED`. The height of the tab's button matrix is calculated from the font height plus padding of the background's and the button's style.

All the listed parts supports the typical background style properties and padding.

`LV_TABVIEW_PART_TAB_BG` has an additional real part, an indicator, called `LV_TABVIEW_PART_INDIC`. It's a thin rectangle-like object under the currently selected tab. When the tab view is animated to an other tab the indicator will be animated too. It can be styles using the typical background style properties. The *size* style property will set the its thickness.

When a new tab is added a *Page* is create for them on `LV_TABVIEW_PART_BG_SCRL` and a new button is added to `LV_TABVIEW_PART_TAB_BG` Button matrix. The created Pages can be used as normal Pages and they have the usual Page parts.

5.31.3 Usage

Adding tab

New tabs can be added with `lv_tabview_add_tab(tabview, "Tab name")`. It will return with a pointer to a *Page* object where the tab's content can be created.

Change tab

To select a new tab you can:

- Click on it on the Button matrix part
- Slide
- Use `lv_tabview_set_tab_act(tabview, id, LV_ANIM_ON/OFF)` function

Change tab's name

To change the name (shown text of the underlying button matrix) of tab `id` during runtime the function `lv_tabview_set_tab_name(tabview, id, name)` can be used.

Tab button's position

By default, the tab selector buttons are placed on the top of the Tab view. It can be changed with `lv_tabview_set_btns_pos(tabview, LV_TABVIEW_TAB_POS_TOP/BOTTOM/LEFT/RIGHT/NONE)`

`LV_TABVIEW_TAB_POS_NONE` will hide the tabs.

Note that, you can't change the tab position from top or bottom to left or right when tabs are already added.

Animation time

The animation time is adjusted by `lv_tabview_set_anim_time(tabview, anim_time_ms)`. It is used when the new tab is loaded.

Scroll propagation

As the tabs' content object is a Page it can receive scroll propagation from an other Page-like object. For example, if a text area is created on the tab's content and that Text area is scrolled but it reached the end the scroll can be propagated to the content Page. It can be enabled with `lv_page/textarea_set_scroll_propagation(obj, true)`.

By default the tab's content Pages have enabled scroll propagation, therefore when they are scrolled horizontally the scroll is propagated to `LV_TABVIEW_PART_BG_SCRL` and this way the Pages will be scrolled.

The manual sliding can be disabled with `lv_page_set_scroll_propagation(tab_page, false)`.

5.31.4 Events

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

- **LV_EVENT_VALUE_CHANGED** Sent when a new tab is selected by sliding or clicking the tab button

Learn more about [Events](#).

5.31.5 Keys

The following *Keys* are processed by the Tabview:

- **LV_KEY_RIGHT/LEFT** Select a tab
- **LV_KEY_ENTER** Change to the selected tab

Learn more about *Keys*.

5.31.6 Example

C

Simple Tabview

code

```
#include "../../lv_examples.h"
#if LV_USE_TABVIEW

void lv_ex_tabview_1(void)
{
    /*Create a Tab view object*/
    lv_obj_t *tabview;
    tabview = lv_tabview_create(lv_scr_act(), NULL);

    /*Add 3 tabs (the tabs are page (lv_page) and can be scrolled*/
    lv_obj_t *tab1 = lv_tabview_add_tab(tabview, "Tab 1");
    lv_obj_t *tab2 = lv_tabview_add_tab(tabview, "Tab 2");
    lv_obj_t *tab3 = lv_tabview_add_tab(tabview, "Tab 3");

    /*Add content to the tabs*/
    lv_obj_t * label = lv_label_create(tab1, NULL);
    lv_label_set_text(label, "This the first tab\n\n"
        "If the content\n"
        "of a tab\n"
        "become too long\n"
        "the it \n"
        "automatically\n"
        "become\n"
        "scrollable.");

    label = lv_label_create(tab2, NULL);
    lv_label_set_text(label, "Second tab");

    label = lv_label_create(tab3, NULL);
    lv_label_set_text(label, "Third tab");
}
#endif
```

MicroPython

No examples yet.

5.32 Text area (lv_textarea)

5.32.1 Overview

The Text Area is a *Page* with a *Label* and a cursor on it. Texts or characters can be added to it. Long lines are wrapped and when the text becomes long enough the Text area can be scrolled.

5.32.2 Parts and Styles

The Text area has the same parts as *Page*. Expect `LV_PAGE_PART_SCRL` because it can't be referenced and it's always transparent. Refer the Page's documentation of details.

Besides the Page parts the virtual `LV_TEXTAREA_PART_CURSOR` part exists to draw the cursor. The cursor's area is always the bounding box of the current character. A block cursor can be created by adding a background color and background opa to `LV_TEXTAREA_PART_CURSOR`'s style. The create line cursor let the cursor transparent and set the *border_side* property.

5.32.3 Usage

Add text

You can insert text or characters to the current cursor's position with:

- `lv_textarea_add_char(textarea, 'c')`
- `lv_textarea_add_text(textarea, "insert this text")`

To add wide characters like 'á', 'ß' or CJK characters use `lv_textarea_add_text(ta, "á")`.

`lv_textarea_set_text(ta, "New text")` changes the whole text.

Placeholder

A placeholder text can be specified - which is displayed when the Text area is empty - with `lv_textarea_set_placeholder_text(ta, "Placeholder text")`

Delete character

To delete a character from the left of the current cursor position use `lv_textarea_del_char(textarea)`. To delete from the right use `lv_textarea_del_char_forward(textarea)`

Move the cursor

The cursor position can be modified directly with `lv_textarea_set_cursor_pos(textarea, 10)`. The `0` position means "before the first characters", `LV_TA_CURSOR_LAST` means "after the last character"

You can step the cursor with

- `lv_textarea_cursor_right(textarea)`
- `lv_textarea_cursor_left(textarea)`
- `lv_textarea_cursor_up(textarea)`
- `lv_textarea_cursor_down(textarea)`

If `lv_textarea_set_cursor_click_pos(textarea, true)` is called the cursor will jump to the position where the Text area was clicked.

Hide the cursor

The cursor can be hidden with `lv_textarea_set_cursor_hidden(textarea, true)`.

Cursor blink time

The blink time of the cursor can be adjusted with `lv_textarea_set_cursor_blink_time(textarea, time_ms)`.

One line mode

The Text area can be configured to be one lined with `lv_textarea_set_one_line(ta, true)`. In this mode the height is set automatically to show only one line, line break character are ignored, and word wrap is disabled.

Password mode

The text area supports password mode which can be enabled with `lv_textarea_set_pwd_mode(textarea, true)`.

If the `•` (Bullet, U+2022) character exists in the font, the entered characters are converted to it after some time or when a new character is entered. If `•` not exists, `*` will be used.

In password mode `lv_textarea_get_text(textarea)` gives the real text, not the bullet characters.

The visibility time can be adjusted with `lv_textarea_set_pwd_show_time(textarea, time_ms)`.

Text align

The text can be aligned to the left, center or right with `lv_textarea_set_text_align(textarea, LV_LABEL_ALIGN_LEFT/CENTER/RIGHT)`.

In one line mode, the text can be scrolled horizontally only if the text is left aligned.

Accepted characters

You can set a list of accepted characters with `lv_textarea_set_accepted_chars(ta, "0123456789.+ -")`. Other characters will be ignored.

Max text length

The maximum number of characters can be limited with `lv_textarea_set_max_length(textarea, max_char_num)`

Very long texts

If there is a very long text in the Text area (e. g. > 20k characters) its scrolling and drawing might be slow. However, by enabling `LV_LABEL_LONG_TXT_HINT 1` in *lv_conf.h* it can be hugely improved. It will save some info about the label to speed up its drawing. Using `LV_LABEL_LONG_TXT_HINT` the scrolling and drawing will be as fast as with "normal" short texts.

Select text

A part of text can be selected if enabled with `lv_textarea_set_text_sel(textarea, true)`. It works like when you select a text on your PC with your mouse.

Scrollbars

The scrollbars can be shown according to different policies set by `lv_textarea_set_scrollbar_mode(textarea, LV_SCROLLBAR_MODE_...)`. Learn more at the [Page](#) object.

Scroll propagation

When the Text area is scrolled on an other scrollable object (like a Page) and the scrolling has reached the edge of the Text area, the scrolling can be propagated to the parent. In other words, when the Text area can be scrolled further, the parent will be scrolled instead.

It can be enabled with `lv_ta_set_scroll_propagation(ta, true)`.

Learn more at the [Page](#) object.

Edge flash

When the Text area is scrolled to edge a circle like flash animation can be shown if it is enabled with `lv_ta_set_edge_flash(ta, true)`

5.32.4 Events

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

- **LV_EVENT_INSERT** Sent when before a character or text is inserted. The event data is the text planned to insert. `lv_ta_set_insert_replace(ta, "New text")` replaces the text to insert. The new text can't be in a local variable which is destroyed when the event callback exists. "" means do not insert anything.
- **LV_EVENT_VALUE_CHANGED** When the content of the text area has been changed.
- **LV_EVENT_APPLY** When `LV_KEY_ENTER` is sent to a text area which is in one line mode.

5.32.5 Keys

- **LV_KEY_UP/DOWN/LEFT/RIGHT** Move the cursor
- **Any character** Add the character to the current cursor position

Learn more about [Keys](#).

5.32.6 Example

C

Simple Text area

code

```
#include "../../lv_examples.h"
#include <stdio.h>
#if LV_USE_TEXTAREA

lv_obj_t * ta1;

static void event_handler(lv_obj_t * obj, lv_event_t event)
{
    if(event == LV_EVENT_VALUE_CHANGED) {
        printf("Value: %s\n", lv_textarea_get_text(obj));
    }
    else if(event == LV_EVENT_LONG_PRESSED_REPEAT) {
        /*For simple test: Long press the Text are to add the text below*/
        const char * txt = "\n\nYou can scroll it if the text is long enough.\n";
        static uint16_t i = 0;
        if(txt[i] != '\0') {
            lv_textarea_add_char(ta1, txt[i]);
            i++;
        }
    }
}

void lv_ex_textarea_1(void)
{
    ta1 = lv_textarea_create(lv_scr_act(), NULL);
    lv_obj_set_size(ta1, 200, 100);
    lv_obj_align(ta1, NULL, LV_ALIGN_CENTER, 0, 0);
}
```

(continues on next page)

(continued from previous page)

```

    lv_textarea_set_text(ta1, "A text in a Text Area");    /*Set an initial text*/
    lv_obj_set_event_cb(ta1, event_handler);
}

#endif

```

Text area with password field

code

```

#include "../../lv_examples.h"
#include <stdio.h>
#if LV_USE_TEXTAREA && LV_USE_KEYBOARD

static void ta_event_cb(lv_obj_t * ta, lv_event_t event);

static lv_obj_t * kb;

void lv_ex_textarea_2(void)
{
    /* Create the password box */
    lv_obj_t * pwd_ta = lv_textarea_create(lv_scr_act(), NULL);
    lv_textarea_set_text(pwd_ta, "");
    lv_textarea_set_pwd_mode(pwd_ta, true);
    lv_textarea_set_one_line(pwd_ta, true);
    lv_textarea_set_cursor_hidden(pwd_ta, true);
    lv_obj_set_width(pwd_ta, LV_HOR_RES / 2 - 20);
    lv_obj_set_pos(pwd_ta, 5, 20);
    lv_obj_set_event_cb(pwd_ta, ta_event_cb);

    /* Create a label and position it above the text box */
    lv_obj_t * pwd_label = lv_label_create(lv_scr_act(), NULL);
    lv_label_set_text(pwd_label, "Password:");
    lv_obj_align(pwd_label, pwd_ta, LV_ALIGN_OUT_TOP_LEFT, 0, 0);

    /* Create the one-line mode text area */
    lv_obj_t * oneline_ta = lv_textarea_create(lv_scr_act(), pwd_ta);
    lv_textarea_set_pwd_mode(oneline_ta, false);
    lv_textarea_set_cursor_hidden(oneline_ta, true);
    lv_obj_align(oneline_ta, NULL, LV_ALIGN_IN_TOP_RIGHT, -5, 20);

    /* Create a label and position it above the text box */
    lv_obj_t * oneline_label = lv_label_create(lv_scr_act(), NULL);
    lv_label_set_text(oneline_label, "Text:");
    lv_obj_align(oneline_label, oneline_ta, LV_ALIGN_OUT_TOP_LEFT, 0, 0);

    /* Create a keyboard */
    kb = lv_keyboard_create(lv_scr_act(), NULL);
    lv_obj_set_size(kb, LV_HOR_RES, LV_VER_RES / 2);

    lv_keyboard_set_textarea(kb, pwd_ta); /* Focus it on one of the text areas to
↪start */
    lv_keyboard_set_cursor_manage(kb, true); /* Automatically show/hide cursors on
↪text areas */
}

```

(continues on next page)

(continued from previous page)

```

}

static void ta_event_cb(lv_obj_t * ta, lv_event_t event)
{
    if(event == LV_EVENT_CLICKED) {
        /* Focus on the clicked text area */
        if(kb != NULL)
            lv_keyboard_set_textarea(kb, ta);
    }

    else if(event == LV_EVENT_INSERT) {
        const char * str = lv_event_get_data();
        if(str[0] == '\n') {
            printf("Ready\n");
        }
    }
}

#endif

```

Text auto-formatting

code

```

#include "../../lv_examples.h"
#include <stdio.h>
#if LV_USE_TEXTAREA && LV_USE_KEYBOARD

static void ta_event_cb(lv_obj_t * ta, lv_event_t event);

static lv_obj_t * kb;

/**
 * Automatically format text like a clock. E.g. "12:34"
 * Add the ':' automatically.
 */
void lv_ex_textarea_3(void)
{
    /* Create the text area */
    lv_obj_t * ta = lv_textarea_create(lv_scr_act(), NULL);
    lv_obj_set_event_cb(ta, ta_event_cb);
    lv_textarea_set_accepted_chars(ta, "0123456789:");
    lv_textarea_set_max_length(ta, 5);
    lv_textarea_set_one_line(ta, true);
    lv_textarea_set_text(ta, "");

    /* Create a keyboard*/
    kb = lv_keyboard_create(lv_scr_act(), NULL);
    lv_obj_set_size(kb, LV_HOR_RES, LV_VER_RES / 2);
    lv_keyboard_set_mode(kb, LV_KEYBOARD_MODE_NUM);
    lv_keyboard_set_textarea(kb, ta);
}

static void ta_event_cb(lv_obj_t * ta, lv_event_t event)
{

```

(continues on next page)

(continued from previous page)

```

    if(event == LV_EVENT_VALUE_CHANGED) {
        const char * txt = lv_textarea_get_text(ta);
        if(txt[0] >= '0' && txt[0] <= '9' &&
           txt[1] >= '0' && txt[1] <= '9' &&
           txt[2] != ':')
        {
            lv_textarea_set_cursor_pos(ta, 2);
            lv_textarea_add_char(ta, ':');
        }
    }
}

#endif

```

MicroPython

No examples yet.

5.33 Tile view (lv_tileview)

5.33.1 Overview

The Tileview is a container object where its elements (called *tiles*) can be arranged in a grid form. By swiping the user can navigate between the tiles.

If the Tileview is screen sized it gives a user interface you might have seen on the smartwatches.

5.33.2 Parts and Styles

The Tileview has the same parts as [Page](#). Expect LV_PAGE_PART_SCROLL because it can't be referenced and it's always transparent. Refer the Page's documentation of details.

5.33.3 Usage

Valid positions

The tiles don't have to form a full grid where every element exists. There can be holes in the grid but it has to be continuous, i.e. there can't be an empty rows or columns.

With `lv_tileview_set_valid_positions(tileview, valid_pos_array, array_len)` the valid positions can be set. Scrolling will be possible only to this positions. The 0,0 index means the top left tile. E.g. `lv_point_t valid_pos_array[] = {{0,0}, {0,1}, {1,1}, {LV_COORD_MIN, LV_COORD_MIN}}` gives a Tile view with "L" shape. It indicates that there is no tile in {1,1} therefore the user can't scroll there.

In other words, the `valid_pos_array` tells where the tiles are. It can be changed on the fly to disable some positions on specific tiles. For example, there can be a 2x2 grid where all tiles are added but the first row ($y = 0$) as a "main row" and the second row ($y = 1$) contains options for the tile above it. Let's say horizontal scrolling is possible only in the main row and not possible between the options in the second row. In this case the `valid_pos_array` needs to be changed when a new main tile is selected:

- for the first main tile: `{0,0}`, `{0,1}`, `{1,0}` to disable the `{1,1}` option tile
- for the second main tile `{0,0}`, `{1,0}`, `{1,1}` to disable the `{0,1}` option tile

Set tile

To set the currently visible tile use `lv_tileview_set_tile_act(tileview, x_id, y_id, LV_ANIM_ON/OFF)`.

Add element

To add elements just create an object on the Tileview and position it manually to the desired position.

`lv_tileview_add_element(tileview, element)` should be used to make possible to scroll (drag) the Tileview by one its element. For example, if there is a button on a tile, the button needs to be explicitly added to the Tileview to enable the user to scroll the Tileview with the button too.

Scroll propagation

The scroll propagation feature of page-like objects (like [List](#)) can be used very well here. For example, there can be a full-sized List and when it reaches the top or bottom most position the user will scroll the tile view instead.

Animation time

The animation time of the Tileview can be adjusted with `lv_tileview_set_anim_time(tileview, anim_time)`.

Animations are applied when

- a new tile is selected with `lv_tileview_set_tile_act`
- the current tile is scrolled a little and then released (revert the original title)
- the current tile is scrolled more than half size and then released (move to the next tile)

Edge flash

An "edge flash" effect can be added when the tile view reached hits an invalid position or the end of tile view when scrolled.

Use `lv_tileview_set_edge_flash(tileview, true)` to enable this feature.

5.33.4 Events

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

- **LV_EVENT_VALUE_CHANGED** Sent when a new tile loaded either with scrolling or `lv_tileview_set_act`. The event data is set to the index of the new tile in `valid_pos_array` (It's type is `uint32_t *`)

5.33.5 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

Learn more about *Keys*.

5.33.6 Example

C

Tileview with content

code

```
#include "../../lv_examples.h"
#if LV_USE_TILEVIEW

void lv_ex_tileview_1(void)
{
    static lv_point_t valid_pos[] = {{0,0}, {0, 1}, {1,1}};
    lv_obj_t *tileview;
    tileview = lv_tileview_create(lv_scr_act(), NULL);
    lv_tileview_set_valid_positions(tileview, valid_pos, 3);
    lv_tileview_set_edge_flash(tileview, true);

    lv_obj_t * tile1 = lv_obj_create(tileview, NULL);
    lv_obj_set_size(tile1, LV_HOR_RES, LV_VER_RES);
    lv_tileview_add_element(tileview, tile1);

    /*Tile1: just a label*/
    lv_obj_t * label = lv_label_create(tile1, NULL);
    lv_label_set_text(label, "Scroll down");
    lv_obj_align(label, NULL, LV_ALIGN_CENTER, 0, 0);

    /*Tile2: a list*/
    lv_obj_t * list = lv_list_create(tileview, NULL);
    lv_obj_set_size(list, LV_HOR_RES, LV_VER_RES);
    lv_obj_set_pos(list, 0, LV_VER_RES);
    lv_list_set_scroll_propagation(list, true);
    lv_list_set_scrollbar_mode(list, LV_SCROLLBAR_MODE_OFF);

    lv_list_add_btn(list, NULL, "One");
    lv_list_add_btn(list, NULL, "Two");
    lv_list_add_btn(list, NULL, "Three");
    lv_list_add_btn(list, NULL, "Four");
    lv_list_add_btn(list, NULL, "Five");
    lv_list_add_btn(list, NULL, "Six");
    lv_list_add_btn(list, NULL, "Seven");
    lv_list_add_btn(list, NULL, "Eight");

    /*Tile3: a button*/
    lv_obj_t * tile3 = lv_obj_create(tileview, tile1);
    lv_obj_set_pos(tile3, LV_HOR_RES, LV_VER_RES);
    lv_tileview_add_element(tileview, tile3);
}
```

(continues on next page)

(continued from previous page)

```

    lv_obj_t * btn = lv_btn_create(tile3, NULL);
    lv_obj_align(btn, NULL, LV_ALIGN_CENTER, 0, 0);
    lv_tileview_add_element(tileview, btn);
    label = lv_label_create(btn, NULL);
    lv_label_set_text(label, "No scroll up");
}

#endif

```

MicroPython

No examples yet.

5.34 Window (lv_win)

5.34.1 Overview

The Window is container-like objects built from a header with title and button and a content area.

5.34.2 Parts and Styles

The main part is `LV_WIN_PART_BG` which holds the two other real parts:

1. `LV_WIN_PART_HEADER`: a header *Container* on the top with a title and control buttons
2. `LV_WIN_PART_CONTENT_SCRL` the scrollable part of a *Page* for the content below the header.

Besides these, `LV_WIN_PART_CONTENT_SCRL` has a scrollbar part called `LV_WIN_PART_CONTENT_SCRL`. Read the documentation of *Page* for more details on the scrollbars.

All parts supports the typical background properties. The title uses the *Text* properties of the header part.

The height of the control buttons is: *header height* - *header padding_top* - *header padding_bottom*.

Title

On the header, there is a title which can be modified by: `lv_win_set_title(win, "New title")`.

Control buttons

Control buttons can be added to the right of the window header with: `lv_win_add_btn_right(win, LV_SYMBOL_CLOSE)`, to add a button to the left side of the window header use `lv_win_add_btn_left(win, LV_SYMBOL_CLOSE)` instead. The second parameter is an *Image* source so it can be a symbol, a pointer to an `lv_img_dsc_t` variable or a path to file.

The width of the buttons can be set with `lv_win_set_btn_width(win, w)`. If `w == 0` the buttons will be square-shaped.

`lv_win_close_event_cb` can be used as an event callback to close the Window.

Scrollbars

The scrollbar behavior can be set by `lv_win_set_scrollbar_mode(win, LV_SCROLLBAR_MODE...)`. See [Page](#) for details.

Manual scroll and focus

To scroll the Window directly you can use `lv_win_scroll_hor(win, dist_px)` or `lv_win_scroll_ver(win, dist_px)`.

To make the Window show an object on it use `lv_win_focus(win, child, LV_ANIM_ON/OFF)`.

The time of scroll and focus animations can be adjusted with `lv_win_set_anim_time(win, anim_time_ms)`

Layout

To set a layout for the content use `lv_win_set_layout(win, LV_LAYOUT...)`. See [Container](#) for details.

5.34.3 Events

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

Learn more about [Events](#).

5.34.4 Keys

The following *Keys* are processed by the Page:

- **LV_KEY_RIGHT/LEFT/UP/DOWN** Scroll the page

Learn more about [Keys](#).

5.34.5 Example

C

Simple window

code

```
#include "../../lv_examples.h"
#if LV_USE_WIN

void lv_ex_win_1(void)
{
    /*Create a window*/
    lv_obj_t * win = lv_win_create(lv_scr_act(), NULL);
    lv_win_set_title(win, "Window title");           /*Set the title*/
}
```

(continues on next page)

(continued from previous page)

```

    /*Add control button to the header*/
    lv_obj_t * close_btn = lv_win_add_btn(win, LV_SYMBOL_CLOSE);           /*Add
↪close button and use built-in close action*/
    lv_obj_set_event_cb(close_btn, lv_win_close_event_cb);
    lv_win_add_btn(win, LV_SYMBOL_SETTINGS);           /*Add a setup button*/

    /*Add some dummy content*/
    lv_obj_t * txt = lv_label_create(win, NULL);
    lv_label_set_text(txt, "This is the content of the window\n\n"
        "You can add control buttons to\n"
        "the window header\n\n"
        "The content area becomes\n"
        "automatically scrollable is it's \n"
        "large enough.\n\n"
        " You can scroll the content\n"
        "See the scroll bar on the right!");
}

#endif

```

MicroPython

No examples yet.

CONTRIBUTING

6.1 Introduction

Join LVGL's community and leave your footprint in the library!

There are a lot of ways to contribute to LVGL even if you are new to the library or even new to programming.

It might be scary to make the first step but you have nothing to be afraid of. A friendly and helpful community is waiting for you. Get to know like-minded people and make something great together.

So let's find which contribution option fits you the best and help you join the development of LVGL!

Before getting started here are some guidelines to make contribution smoother:

- Be kind and friendly.
- Be sure to read the relevant part of the documentation before posting a question.
- Ask questions in the [Forum](#) and use [GitHub](#) for development-related discussions.
- Always fill out the post or issue templates in the Forum or GitHub (or at least provide equivalent information). It makes much easier to understand your case and you will get a useful answer faster.
- If possible send an absolute minimal but buildable code example in order to reproduce the issue. Be sure it contains all the required variable declarations, constants, and assets (images, fonts).
- Use [Markdown](#) to format your posts. You can learn it in 10 minutes.
- Speak about one thing in one issue or topic. It makes your post easier to find later for someone with the same question.
- Give feedback and close the issue or mark the topic as solved if your question is answered.
- For non-trivial fixes and features, it's better to open an issue first to discuss the details instead of sending a pull request directly.
- Please read and follow the Coding style guide.

6.2 Pull request

Merging new code into lvgl, documentation, blog, examples, and other repositories happen via *Pull requests* (PR for short). A PR is a notification like "Hey, I made some updates to your project. Here are the changes, you can add them if you want." To do this you need a copy (called fork) of the original project under your account, make some changes there, and notify the original repository about your updates. You can see how it looks like on GitHub for lvgl here: <https://github.com/lvgl/lvgl/pulls>.

To add your changes you can edit files online on GitHub and send a new Pull request from there (recommended for small changes) or add the updates in your favorite editor/IDE and use git to publish the changes (recommended for more complex updates).

6.2.1 From GitHub

1. Navigate to the file you want to edit.
2. Click the Edit button in the top right-hand corner.
3. Add your changes to the file
4. Add a commit message on the bottom of the page
5. Click the *Propose changes* button

6.2.2 From command line

The instructions describe the main **lvgl** repository but it works the same way for the other repositories.

1. Fork the **lvgl repository**. To do this click the "Fork" button in the top right corner. It will "copy" the **lvgl** repository to your GitHub account (https://github.com/<YOUR_NAME>?tab=repositories)
2. Clone your forked repository.
3. Add your changes. You can create a *feature branch* from *master* for the updates: `git checkout -b the-new-feature`
4. Commit and push you changed to the forked **lvgl** repository.
5. Create a PR on GitHub from the page of your **lvgl** repository (https://github.com/<YOUR_NAME>/lvgl) by clicking the "New pull request" button. Don't forget to select the branch where you added your changes.
6. Set the base branch. It means where you want to merge your update. In the **lvgl** repo fixes go to **master**, new features to **dev** branch.
7. Describe what is in the update. An example code is welcome if applicable.
8. If you need to make more changes, just update your forked **lvgl** repo with new commits. They will automatically appear in the PR.

6.3 Developer Certification of Origin (DCO)

6.3.1 Overview

To ensure that all licensing criteria is met for all repositories of the LVGL project we apply a process called DCO (Developer's Certificate of Origin).

The text of DCO can be read here: <https://developercertificate.org/>.

By contributing to any repositories of the LVGL project you state that your contribution corresponds with the DCO.

No further action is required if your contribution fulfills the DCO. If you are not sure about it feel free to ask us in a comment.

6.3.2 Accepted licenses and copyright notices

To make the DCO easier to digest, here are some practical guides about specific cases:

Your own work

The simplest case is when the contribution is solely your own work. In this case you can just send a Pull Request without worrying about any licensing issues.

Use code from online source

If the code you would like to add is based on an article, post or comment on a website (e.g. StackOverflow) the license and/or rules of that site should be followed.

For example in case of StackOverflow a notice like this can be used:

```
/* The original version of this code-snippet was published on StackOverflow.
 * Post: http://stackoverflow.com/questions/12345
 * Author: http://stackoverflow.com/users/12345/username
 * The following parts of the snippet were changed:
 * - Check this or that
 * - Optimize performance here and there
 */
... code snippet here ...
```

Use MIT licensed code

As LVGL is also MIT licensed other MIT licensed code can be integrated without issues. The MIT license requests a copyright notice be added to the derived work. So you need to copy the original work's license file or it's text to the code you want to add.

Use GPL licensed code

As GPL license is not compatible with MIT license so LVGL can not accept GPL licensed code.

6.4 When you get started with LVGL

Even if you're just getting started with LVGL there are plenty of ways to get your feet wet. Most of these options don't even require knowing a single line of code of LVGL.

6.4.1 Give LVGL a Star

Show that you like LVGL by giving it star on GitHub!

Star

This simple click makes LVGL more visible on GitHub and makes it more attractive to other people. So with this, you already helped a lot!

6.4.2 Tell what you have achieved

Have you already started using LVGL in a *Simulator*, a development board, or your custom hardware? Was it easy or were there some obstacles? Are you happy with the result?

If so why don't you tell it to your friends? You can post it on Twitter, Facebook, LinkedIn, or create a YouTube video.

Any of these helps a lot to spread the word of LVGL and familiarize it with new developers.

Only thing: don't forget to add a link to <https://lvgl.io> or <https://github.com/lvgl> and [#lvgl](#). Thank you! :)

6.4.3 Write examples

As you learn LVGL probably you will play with the features of widgets. But why don't you publish your experiments?

Every widgets' documentation contains some examples. For example here are the examples of the *Drop-down list*. The examples are directly loaded from the [lv_examples](#) repository.

So all you need to do is send a *Pull request* to the [lv_examples](#) repository and follow some conventions:

- Name the examples like `lv_ex_<widget_name>_<id>`
- Make the example as short and simple as possible
- Add comments to explain what the example does
- Use 320x240 resolution
- Create a screenshot about the example
- Update `index.rst` in the example's folder with your new example. See how the other examples are added.

6.4.4 Improve the docs

As you read the documentation you might see some typos or unclear sentences. For typos and straightforward fixes, you can simply edit the file on GitHub. There is an **Edit on Github** link on the top right-hand corner of all pages. Click it to see the file on GitHub, hit the Edit button, and add your fixes as described in *Pull request - From Github* section.

Note that the documentation is also formatted in [Markdown](#).

6.4.5 Translate the docs

If you have more free time you can even translate the documentation. The currently available languages are shown in the [locals](#) folder.

If your chosen language is still not added, please write a [comment here](#).

To add your translations:

- Find the *.po* in `<language_code>/LC_MESSAGES/<section_name>.po`. E.g. the widgets translated to German should be in `de/LC_MESSAGES/widgets.po`.
- Open a po file and fill the `msgstr` fields with the translation
- Send a *Pull request*

To display a translation in the public documentation page at least these sections should be translated:

- Get started: Quick overview
- Overview: Objects, Events, Styles
- Porting: System overview, Set-up a project, Display interface, Input device Interface, Tick interface
- 5 widgets of your choice

6.4.6 Write a blog post

The [LVGL Blog](#) welcomes posts from anyone. It's a good place to talk about a project you created with LVGL, write a tutorial, or share some nice tricks. The latest blog posts are shown on the [homepage of LVGL](#) to make your work more visible.

The blog is hosted on GitHub. If you add a post GitHub automatically turns it into a website. See the [README](#) of the blog repo to see how to add your post.

6.5 When you already use LVGL

6.5.1 Give feedback

Let us know what you are working on! You can open a new topic in the [My projects](#) category of the Forum. Showing your project to others is a win-win situation because it increases your and LVGL's reputation at the same time.

If you don't want to speak about it publicly feel free to use [Contact form](#) on [lvgl.io](#) to private message to us.

6.5.2 Report bugs

As you use LVGL you might find bugs. Before reporting them be sure to check the relevant parts of the documentation.

If it really seems like a bug feel free to open an [issue on GitHub](#).

When filing the issue be sure to fill the template. It helps a lot to find the root of the problems and helps to avoid a lot of questions.

6.5.3 Send fixes

The beauty of open-source software is you can easily dig in to it to understand how it works. You can also fix or adjust it as you wish.

If you found and fixed a bug don't hesitate to send a *Pull request* with the fix.

In your Pull request please also add a line to `CHANGELOG.md`.

6.5.4 Join the conversations in the Forum

It feels great to know you are not alone if something is not working. It's even better to help others when they struggle with something.

While you were learning LVGL you might have questions and used the Forum to get answers. As a result, you probably have more knowledge about how LVGL works.

One of the best ways to give back is to use the Forum and answer the questions of newcomers - like you were once.

Just read the titles and if you are familiar with the topic don't hesitate to share your thoughts and suggestions.

Participating in the discussions is one of the best ways to become part of the project and get to know like-minded people!

6.5.5 Add features

We collect the planned features in GitHub issues tracker and mark them with [Help wanted](#) label. If you are interested in any of them feel free to share your opinion and/or participate in the the implementation.

Other features which are (still) not on the road map are listed in the [Feature request](#) category of the Forum. If you have a feature idea for LVGL please use the Forum to share it! Make sure to check that there isn't an existing post; if there is, you should comment on it instead to show that there is increased interest in an existing request.

When adding a new features the followings also needs to be updated:

- Add a line to `CHANGELOG.md`.
- Update the documentation. See this [guide](#).
- Add an example if applicable. See this [guide](#).

6.6 When you are confident with LVGL

6.6.1 Become a maintainer

If you want to become part of the core development team, you can become a maintainer of a repository.

By becoming a maintainer:

- you get write access to that repo: - add code directly without sending a pull request - accept pull requests - close/reopen/edit issues
- your name will be added in the credits section of lvgl.io/about (will be added soon) and lvgl's README.
- you can join the [Core_contributor](#) group in the Forum and get the LVGL logo on your avatar.
- your word has higher impact when we make decisions

You can become a maintainer by invitation, however the following conditions need to met

1. Have > 50 replies in the Forum. You can look at your stats [here](#)
2. Send > 5 non-trivial pull requests to the repo where you would like to be a maintainer

If you are interested, just send a message (e.g. from the Forum) to the current maintainers of the repository. They will check if the prerequisites are met. Note that meeting the prerequisites is not a guarantee of acceptance, i.e. if the conditions are met you won't automatically become a maintainer. It's up to the current maintainers to make the decision.

6.6.2 Move your project repository under LVGL organization

Besides the core `lvgl` repository there are other repos for ports to development boards, IDEs or other environment. If you ported LVGL to a new platform we can host it under the LVGL organization among the other repos.

This way your project will become part of the whole LVGL project and can get more visibility. If you are interested in this opportunity just open an [issue in lvgl repo](#) and tell what you have!

If we agree that your port is useful, we will open a repository for your project where you will have admin rights.

To make this concept sustainable there are a few rules to follow:

- You need to add a README to your repo.
- We expect to maintain the repo to some extent:
 - Follow at least the major versions of lvgl
 - Respond to the issues (in a reasonable time)
- If there is no activity in a repo for 6 months it will be archived