

---

# **LVGL\_Chinese\_Documents**

**LVGL community & geekfuns**

2022 年 01 月 05 日

<b>1</b>	<b>Welcome to the documentation of LVGL!</b>	<b>2</b>
1.1	介绍 (Introduction)	2
1.2	快速开始 (Get started)	7
1.3	移植 (Porting)	28
1.4	概览 (Overview)	42
1.5	部件 (Widgets)	123
1.6	布局 (Layouts)	189
1.7	第三方库 (3rd party libraries)	195
1.8	其他 (Others)	204
1.9	贡献 (Contributing)	206
1.10	更新日志 (Changelog)	213
1.11	后续目标 (Roadmap)	246

这是我在学习 LVGL 的过程中顺手翻译的作品，若有纰漏请多包涵  
另外，开头介绍性的文字由于重要程度较低，我将选择性翻译或者直接机翻

---

Welcome to the documentation of LVGL!

---

## 1.1 介绍 (Introduction)

LVGL (Light and Variables Graphics Library) 是一个免费的开源图形库，提供了创建具有易于使用的图形元素、优美的视觉效果和低内存占用的嵌入式 GUI 所需的一切。

### 1.1.1 特性 (Key features)

- 功能强大的构建块，如按钮、图表、列表、滑块、图像等。
- 具有动画、抗锯齿、不透明度和平滑滚动的高级图形
- 各种输入设备，如触摸板、鼠标、键盘、编码器等。
- 多语言支持 UTF-8 编码
- 多显示器支持，即同时使用多个 TFT 单色显示器
- 具有类似 CSS 样式的完全可自定义图形元素
- 硬件独立：与任何微控制器或显示器一起使用
- 可扩展性：能够使用少量内存 (64KB 闪存、16KB RAM) 运行
- 支持操作系统、外部内存和 GPU，但不是必需的
- 单帧缓冲操作，即使具有高级图形效果
- 用 C 编写以实现最大兼容性 (与 C++ 兼容)
- 在没有嵌入式硬件的 PC 上启动嵌入式 GUI 设计的模拟器
- 绑定到 MicroPython
- 快速 GUI 设计的教程、示例和主题
- 文档可在线或以 PDF 格式获取
- 麻省理工学院许可下的免费开源

### 1.1.2 软硬件需求 (Requirements)

基本上，每个能够驱动显示器的现代控制器都适合运行 LVGL。最低要求是：

### 1.1.3 许可 (License)

The LVGL project (including all repositories) is licensed under [MIT license](#). This 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 to [lvgl.io](#).

Although you can get LVGL for free there is a massive amount of 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 different ways of contributing* such as simply writing a tweet about you are using LVGL, fixing bugs, translating the documentation, or even becoming a maintainer.

### 1.1.4 仓库结构 (Repository layout)

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

You will find these repositories there:

- [lvgl](#) The library itself with many [examples](#).
- [lv\\_demos](#) Demos created with LVGL.
- [lv\\_drivers](#) Display and input device drivers
- [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

### 1.1.5 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

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

## Release cycle

- Bug fixes: Released on demand even weekly
- Minor releases: Every 3-4 months
- Major releases: Approximately yearly

## Branches

The core repositories have at least the following branches:

- `master` latest version, patches are merged directly here.
- `release/vX.Y` stable versions of the minor releases
- `fix/some-description` temporary branches for bug fixes
- `feat/some-description` temporary branches for features

## Changelog

The changes are recorded in *CHANGELOG.md*.

## Version support

Before v8 every minor release of major releases is supported for 1 year. Starting from v8, every minor release is supported for 1 year.

### 1.1.6 FAQ

#### Where can I ask questions?

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

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

#### 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.

This includes:

- “Common” MCUs like STM32F, STM32H, NXP Kinetis, LPC, iMX, dsPIC33, PIC32 etc.
- Bluetooth, GSM, Wi-Fi modules like Nordic NRF and Espressif ESP32
- Linux with frame buffer device such as `/dev/fb0`. This includes Single-board computers like the Raspberry Pi
- Anything else with a strong enough MCU and a peripheral to drive a display

## 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 it with LVGL.

Some examples of the supported display types:

- TFTs with 16 or 24 bit color depth
- Monitors with an 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.

## 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_timer_handler()` in your main `while(1)`.

Learn more in the [Tick](#) and [Task handler](#) sections.

## Why is the display driver 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*” .

## Why do 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);
```

### Why do I see nonsense colors on the screen?

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

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

### How to speed up my UI?

- Turn on compiler optimization and enable cache if your MCU has it
- Increase the size of the display buffer
- Use two display buffers and flush the buffer with DMA (or similar peripheral) 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 a SPI port consider changing to a model with a parallel interface because it has much higher throughput
- Keep the display buffer in internal RAM (not in external SRAM) because LVGL uses it a lot and it should have a fast access time

### 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 and `-gc-sections` linker flag to remove unused functions and variables from the final binary.

### How to reduce the RAM usage

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

### How to work with an operating system?

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



## 1.2 快速开始 (Get started)

There are several ways to get your feet wet with LVGL. Here is one recommended order of documents to read and things to play with when you are learning to use LVGL:

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. Check out the Platform-specific tutorials. (in this section below). (10 minutes)
7. Port LVGL to a board. See the [Porting](#) guide or check the ready to use [Projects](#)
8. Read the [Overview](#) page to get a better understanding of the library. (2-3 hours)
9. Check the documentation of the [Widgets](#) to see their features and usage
10. If you have questions got to the [Forum](#)
11. Read the [Contributing](#) guide to see how you can help to improve LVGL (15 minutes)

### 1.2.1 快速概览 Quick overview

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

#### Get started in a simulator

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

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

#### Add LVGL into your project

If you would rather try LVGL on your own project follow these steps:

- [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_template.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 the `LV_COLOR_DEPTH` defines.
- Include `lvgl/lvgl.h` in files 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. Alternatively, configure `LV_TICK_CUSTOM` (see `lv_conf.h`) so that LVGL can retrieve the current time directly.
- Call `lv_init()`

- Create a draw buffer: 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_draw_buf_t draw_buf;
static lv_color_t buf1[DISP_HOR_RES * DISP_VER_RES / 10];
/*Declare a buffer for 1/10 screen size*/
lv_disp_draw_buf_init(&draw_buf, buf1, NULL, MY_DISP_HOR_RES * MY_DISP_VER_RES / 10);
/*Initialize the display buffer.*/
```

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

```
static 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.draw_buf = &draw_buf;          /*Assign the buffer to the display*/
disp_drv.hor_res = MY_DISP_HOR_RES;     /*Set the horizontal resolution of the display*/
disp_drv.ver_res = MY_DISP_VER_RES;     /*Set the vertical resolution of 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;
    /*It's a very slow but simple implementation.
    *`set_pixel` needs to be written by you to set pixel on the screen*/
    for(y = area->y1; y <= area->y2; y++) {
        for(x = area->x1; x <= area->x2; x++) {
            set_pixel(x, y, *color_p);
            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 touchpad:

```
static 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*/

void my_touchpad_read(lv_indev_t * indev, lv_indev_data_t * data)
{
    /*`touchpad_is_pressed` and `touchpad_get_xy` needs to be implemented by you*/
    if(touchpad_is_pressed()) {
        data->state = LV_INDEV_STATE_PRESSED;
        touchpad_get_xy(&data->point.x, &data->point.y);
    } else {
        data->state = LV_INDEV_STATE_RELEASED;
    }
}
```

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

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

## Learn the basics

### Widgets

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

Every object has a parent object where it is created. 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 within their parent's bounding area. In other words, the parts of the children outside 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(screen)`.

You can create a new object with `lv_<type>_create(parent)`. It will return an `lv_obj_t *` variable that can be used as a reference to the object to set its parameters.

For example:

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

To set some basic attributes `lv_obj_set_<parameter_name>(obj, <value>)` functions can be used. For example:

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

Along with the basic attributes, widgets can have type specific parameters which are set by `lv_<widget_type>_set_<parameter_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/widgets/lv_slider.h`).

### Events

Events are used to inform the user that something has happened with an object. You can assign one or more callbacks to an object which will be called if the object is clicked, released, dragged, being deleted, etc.

A callback is assigned like this:

```
lv_obj_add_event_cb(btn, btn_event_cb, LV_EVENT_CLICKED, NULL); /*Assign a callback
↳ to the button*/

...

void btn_event_cb(lv_event_t * e)
{
    printf("Clicked\n");
}
```

LV\_EVENT\_ALL can be used instead of LV\_EVENT\_CLICKED to invoke the callback for any event.

From `lv_event_t * e` the current event code can be retrieved with:

```
lv_event_code_t code = lv_event_get_code(e);
```

The object that triggered the event can be retrieved with:

```
lv_obj_t * obj = lv_event_get_target(e);
```

To learn all features of the events go to the [Event overview](#) section.

## Parts

Widgets might be built from one or more *parts*. For example, a button has only one part called LV\_PART\_MAIN. However, a *Slider* has LV\_PART\_MAIN, LV\_PART\_INDICATOR and LV\_PART\_KNOB.

By using parts you can apply different styles to sub-elements of a widget. (See below)

Read the widgets' documentation to learn which parts each uses.

## States

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

- LV\_STATE\_DEFAULT Normal, released state
- LV\_STATE\_CHECKED Toggled or checked state
- LV\_STATE\_FOCUSED Focused via keypad or encoder or clicked via touchpad/mouse
- LV\_STATE\_FOCUS\_KEY Focused via keypad or encoder but not via touchpad/mouse
- LV\_STATE\_EDITED Edit by an encoder
- LV\_STATE\_HOVERED Hovered by mouse (not supported now)
- LV\_STATE\_PRESSED Being pressed
- LV\_STATE\_SCROLLED Being scrolled
- LV\_STATE\_DISABLED Disabled

For example, if you press an object it will automatically go to the LV\_STATE\_FOCUSED and LV\_STATE\_PRESSED states and when you release it the LV\_STATE\_PRESSED state will be removed while focus remains active.

To check if an object is in a given state use `lv_obj_has_state(obj, LV_STATE_...)`. It will return `true` if the object is currently in that state.

To manually add or remove states use:

```
lv_obj_add_state(obj, LV_STATE_...);
lv_obj_clear_state(obj, LV_STATE_...);
```

## Styles

A style instance contains properties such as background color, border width, font, etc. that describe the appearance of objects.

Styles are represented with `lv_style_t` variables. Only their pointer is saved in the objects so they need to be defined as static or global. Before using a style it needs to be initialized with `lv_style_init(&style1)`. After that, properties can be added to configure the style. For example:

```
static lv_style_t style1;
lv_style_init(&style1);
lv_style_set_bg_color(&style1, lv_color_hex(0xa03080))
lv_style_set_border_width(&style1, 2))
```

See the full list of properties [here](#).

Styles are assigned using the ORed combination of an object's part and state. For example to use this style on the slider's indicator when the slider is pressed:

```
lv_obj_add_style(slider1, &style1, LV_PART_INDICATOR | LV_STATE_PRESSED);
```

If the *part* is `LV_PART_MAIN` it can be omitted:

```
lv_obj_add_style(btn1, &style1, LV_STATE_PRESSED); /*Equal to LV_PART_MAIN | LV_STATE_PRESSED*/
```

Similarly, `LV_STATE_DEFAULT` can be omitted too:

```
lv_obj_add_style(slider1, &style1, LV_PART_INDICATOR); /*Equal to LV_PART_INDICATOR | LV_STATE_DEFAULT*/
```

For `LV_STATE_DEFAULT` and `LV_PART_MAIN` simply write 0:

```
lv_obj_add_style(btn1, &style1, 0); /*Equal to LV_PART_MAIN | LV_STATE_DEFAULT*/
```

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 the background color to make the button red:

```
lv_obj_add_style(btn1, &style_btn, 0);
lv_obj_add_style(btn1, &style1_btn_red, 0);
```

If a property is not set on for the current state, the style with `LV_STATE_DEFAULT` will be used. A default value is used if the property is not defined in the default state.

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

Local style properties also can be added to objects. This creates a style which resides inside the object and is used only by the object:

```
lv_obj_set_style_bg_color(slider1, lv_color_hex(0x2080bb), LV_PART_INDICATOR | LV_STATE_PRESSED);
```

To learn all the features of styles see the [Style overview](#) section.

## Themes

Themes are the default styles for objects. Styles from a theme are applied automatically when objects are created.

The theme for your application is a compile time configuration set in `lv_conf.h`.

## Examples

### 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)
```

## 1.2.2 Simulator on PC

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

Using the simulator on a PC has the following advantages:

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

## 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.

- [Eclipse with SDL driver](#): Recommended on Linux and Mac
- [CodeBlocks](#): Recommended on Windows
- [VisualStudio with SDL driver](#): For Windows
- [VSCode with SDL driver](#): Recommended on Linux and Mac
- [PlatformIO with SDL driver](#): Recommended on Linux and Mac

You can use any IDE for development but, for simplicity, the configuration for Eclipse CDT is what we'll focus on in this tutorial. The following section describes the set-up guide of Eclipse CDT in more detail.

**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.

## Set-up Eclipse CDT

### Install Eclipse CDT

Eclipse CDT is a C/C++ IDE.

Eclipse is a Java-based tool so 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 install a '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 touchpad.

## 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 LVGL 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 cases the configuration in the downloaded project is enough.

After a successful 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 you are ready to use LVGL and begin development on your PC.

### 1.2.3 STM32

TODO

### 1.2.4 NXP

NXP has integrated LVGL into the MCUXpresso SDK packages for several of their general purpose and crossover microcontrollers, allowing easy evaluation and migration into your product design. [Download an SDK for a supported board](#) today and get started with your next GUI application.



## Creating new project with LVGL

Downloading the MCU SDK example project is recommended as a starting point. It comes fully configured with LVGL (and with PXP support if module is present), no additional integration work is required.

## Adding HW acceleration for NXP iMX RT platforms using PXP (PiXeL Pipeline) engine for existing projects

Several drawing features in LVGL can be offloaded to the PXP engine. The CPU is available for other operations while the PXP is running. An RTOS is required to block the LVGL drawing thread and switch to another task or suspend the CPU for power savings.

### Features supported:

- RGB565 color format
- Area fill + optional transparency
- BLIT (BLock Image Transfer) + optional transparency
- Color keying + optional transparency
- Recoloring (color tint) + optional transparency
- RTOS integration layer
- Default FreeRTOS and bare metal code provided

### Basic configuration:

- Select NXP PXP engine in lv\_conf.h: Set LV\_USE\_GPU\_NXP\_PXP to 1
- Enable default implementation for interrupt handling, PXP start function and automatic initialization: Set LV\_USE\_GPU\_NXP\_PXP\_AUTO\_INIT to 1
- If FSL\_RTOS\_FREE\_RTOS symbol is defined, FreeRTOS implementation will be used, otherwise bare metal code will be included

### Basic initialization:

- If LV\_USE\_GPU\_NXP\_PXP\_AUTO\_INIT is enabled, no user code is required; PXP is initialized automatically in lv\_init()
- For manual PXP initialization, default configuration structure for callbacks can be used. Initialize PXP before calling lv\_init()

```
#if LV_USE_GPU_NXP_PXP
    #include "lv_gpu/lv_gpu_nxp_pxp.h"
    #include "lv_gpu/lv_gpu_nxp_pxp_osa.h"
#endif
. . .
#if LV_USE_GPU_NXP_PXP
    if (lv_gpu_nxp_pxp_init(&pxp_default_cfg) != LV_RES_OK) {
        PRINTF("PXP init error. STOP.\n");
        for ( ; ; ) ;
    }
}
```

(下页继续)

(续上页)

```

}
#endif

```

### Project setup:

- Add PXP related files to project:
  - lv\_gpu/lv\_gpu\_nxp.c, lv\_gpu/lv\_gpu\_nxp.h: low level drawing calls for LVGL
  - lv\_gpu/lv\_gpu\_nxp\_osa.c, lv\_gpu/lv\_gpu\_osa.h: default implementation of OS-specific functions (bare metal and FreeRTOS only)
    - \* optional, required only if LV\_USE\_GPU\_NXP\_PXP\_AUTO\_INIT is set to 1
- PXP related code depends on two drivers provided by MCU SDK. These drivers need to be added to project:
  - fsl\_pxp.c, fsl\_pxp.h: PXP driver
  - fsl\_cache.c, fsl\_cache.h: CPU cache handling functions

### Advanced configuration:

- Implementation depends on multiple OS-specific functions. The struct `lv_nxp_pxp_cfg_t` with callback pointers is used as a parameter for the `lv_gpu_nxp_pxp_init()` function. Default implementation for FreeRTOS and baremetal is provided in `lv_gpu_nxp_osa.c`
  - `pxp_interrupt_init()`: Initialize PXP interrupt (HW setup, OS setup)
  - `pxp_interrupt_deinit()`: Deinitialize PXP interrupt (HW setup, OS setup)
  - `pxp_run()`: Start PXP job. Use OS-specific mechanism to block drawing thread. PXP must finish drawing before leaving this function.
- There are configurable area thresholds which are used to decide whether the area will be processed by CPU, or by PXP. Areas smaller than a defined value will be processed by CPU and those bigger than the threshold will be processed by PXP. These thresholds may be defined as preprocessor variables. Default values are defined in `lv_gpu/lv_gpu_nxp_pxp.h`
  - `GPU_NXP_PXP_BLIT_SIZE_LIMIT`: size threshold for image BLIT, BLIT with color keying, and BLIT with recolor ( $\text{OPA} > \text{LV\_OPA\_MAX}$ )
  - `GPU_NXP_PXP_BLIT_OPA_SIZE_LIMIT`: size threshold for image BLIT and BLIT with color keying with transparency ( $\text{OPA} < \text{LV\_OPA\_MAX}$ )
  - `GPU_NXP_PXP_FILL_SIZE_LIMIT`: size threshold for fill operation ( $\text{OPA} > \text{LV\_OPA\_MAX}$ )
  - `GPU_NXP_PXP_FILL_OPA_SIZE_LIMIT`: size threshold for fill operation with transparency ( $\text{OPA} < \text{LV\_OPA\_MAX}$ )

### 1.2.5 Espressif (ESP32)

Since v7.7.1 LVGL includes a Kconfig file, so LVGL can be used as an ESP-IDF v4 component.

#### Get the LVGL demo project for ESP32

We've created `lv_port_esp32`, a project using ESP-IDF and LVGL to show one of the demos from `lv_examples`. You are able to configure the project to use one of the many supported display controllers, see `lvgl_esp32_drivers` for a complete list of supported display and indev (touch) controllers.

#### Use LVGL in your ESP32 project

##### Prerequisites

ESP-IDF v4 framework is the suggested version to use.

##### Get LVGL

It is suggested that you add LVGL as a “component” to your project. This component can be located inside a directory named “components” in the project root directory.

When your project is a git repository you can include LVGL as a git submodule:

```
git submodule add https://github.com/lvgl/lvgl.git components/lvgl
```

The above command will clone LVGL's main repository into the `components/lvgl` directory. LVGL includes a `CMakeLists.txt` file that sets some configuration options so you can use LVGL right away.

When you are ready to configure LVGL, launch the configuration menu with `idf.py menuconfig` on your project root directory, go to **Component config** and then **LVGL configuration**.

#### Use `lvgl_esp32_drivers` in your project

You can also add `lvgl_esp32_drivers` as a “component”. This component can be located inside a directory named “components” on your project root directory.

When your project is a git repository you can include `lvgl_esp32_drivers` as a git submodule:

```
git submodule add https://github.com/lvgl/lvgl\_esp32\_drivers.git components/lvgl_esp32_drivers
```

#### Support for ESP32-S2

Basic support for ESP32-S2 has been added into the `lvgl_esp32_drivers` repository.

## 1.2.6 Arduino

The [core LVGL library](#) and the [demos](#) 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.

### Get the LVGL Arduino library

LVGL can be installed via the Arduino IDE Library Manager or as a .ZIP library.

### 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.

### Configure LVGL

LVGL has its own configuration file called `lv_conf.h`. When LVGL is installed, follow these configuration steps:

1. Go to directory of the installed Arduino libraries
2. Go to `lvgl` and copy `lv_conf_template.h` as `lv_conf.h` into the Arduino Libraries directory next to the `lvgl` library folder.
3. Open `lv_conf.h` and change the first `#if 0` to `#if 1`
4. Set the color depth of you display in `LV_COLOR_DEPTH`
5. Set `LV_TICK_CUSTOM 1`

### Initialize LVGL and run an example

Take a look at [LVGL\\_Arduino.ino](#) to see how to initialize LVGL. [TFT\\_eSPI](#) is used as the display driver.

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

Note that, there is no dedicated INO file for every example, but you can open the examples in `lvgl/examples` folder and copy-paste them to your INO file. You can NOT call the examples like `lv_example_btn_1()` because the Arduino doesn't compile the examples.

You can the [lv\\_demos](#) library which needs to be installed and configured separately.

## Debugging and logging

LVGL can display debug information in case of trouble. In the `LVGL_Arduino.ino` example there is a `my_print` method, which sends this debug information to the serial interface. To enable this feature you have to edit the `lv_conf.h` file and enable logging in the 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 the log module and setting `LV_LOG_LEVEL` accordingly, the output log is sent to the `Serial` port @ 115200 bps.

## 1.2.7 Micropython

### 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.)

## 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.
  - Shortening the cycle of changing and fine-tuning the GUI.
  - Modelling 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 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).
- 

## 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)
```

## 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

In 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 chances are you would want to create your own input/display drivers for your specific hardware. Drivers can be implemented either in C as a Micropython module, or in pure Micropython!

### Where can I find more information?

- In this [Blog Post](#)
- [lv\\_micropython README](#)
- [lv\\_binding\\_micropython README](#)
- The [LVGL micropython forum](#) (Feel free to ask anything!)
- At Micropython: [docs](#) and [forum](#)

## 1.2.8 Tasmota and berry

### What is Tasmota?

[Tasmota](#) is a widely used open-source firmware for ESP8266 and ESP32 based devices. It supports a wide variety of devices, sensors and integrations to Home Automation and Cloud services. Tasmota firmware is downloaded more than 200,000 times each month, and has an active and growing community.

Tasmota provides access to hundreds of supported devices, full support of MQTT, HTTP(S), integration with major Home Automation systems, myriad of sensors, IR, RF, Zigbee, Bluetooth, AWS IoT, Azure IoT, Alexa and many more.

## What is Berry?

**Berry** is a ultra-lightweight dynamically typed embedded scripting language. It is designed for lower-performance embedded devices. The interpreter of Berry include a one-pass compiler and register-based VM, all the code is written in ANSI C99. Berry offers a syntax very similar to Python, and is inspired from LUA VM. It is fully integrated in Tasmota

## Highlights of Berry

Berry has the following advantages:

- **Lightweight:** A well-optimized interpreter with very little resources. Ideal for use in microprocessors.
- **Fast:** optimized one-pass bytecode compiler and register-based virtual machine.
- **Powerful:** supports imperative programming, object-oriented programming, functional programming.
- **Flexible:** Berry is a dynamic type script, and it' s intended for embedding in applications. It can provide good dynamic scalability for the host system.
- **Simple:** simple and natural syntax, support garbage collection, and easy to use FFI (foreign function interface).
- **RAM saving:** With compile-time object construction, most of the constant objects are stored in read-only code data segments, so the RAM usage of the interpreter is very low when it starts.

All features are detailed in the [Berry Reference Manual](#)

---

## Why LVGL + Tasmota + Berry?

In 2021, Tasmota added full support of LVGL for ESP32 based devices. It also introduced the Berry scripting language, a small-footprint language similar to Python and fully integrated in Tasmota.

A comprehensive mapping of LVGL in Berry language is now available, similar to the mapping of Micropython. It allows to use +98% of all LVGL features. It is also possible to write custom widgets in Berry.

Versions supported: LVGL v8.0.2, LodePNG v20201017, Freetype 2.10.4

## Tasmota + Berry + LVGL could be used for:

- Fast prototyping GUI.
- Shortening the cycle of changing and fine-tuning the GUI.
- Modelling the GUI in a more abstract way by defining reusable composite objects, taking advantage of Berry' s language features such as Inheritance, Closures, Exception Handling...
- Make LVGL accessible to a larger audience. No need to know C to create a nice GUI on an embedded system.

A higher level interface compatible with [OpenHASP](#) is also under development.

---



## So what does it look like?

TL;DR: Similar to MicroPython, it's very much like the C API, but Object-Oriented for LVGL components.

Let's dive right into an example!

## A simple example

```
lv.start()           # start LVGL
scr = lv.scr_act()   # get default screen
btn = lv.btn(scr)    # create button
btn.center()
label = lv.label(btn) # create a label in the button
label.set_text("Button") # set a label to the button
```

## How can I use it?

You can start in less than 10 minutes on a M5Stack or equivalent device in less than 10 minutes in this [short tutorial](#)

## Where can I find more information?

### 1.2.9 NuttX RTOS

#### 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 microcontrollers and microprocessors and compliant 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 to think of it as a small Unix/Linux for microcontrollers.

#### Highlights of NuttX

- **Small** - Fits and runs in microcontrollers as small as 32 kB Flash and 8 kB 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 allows 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, so you can use it to create predictable applications for realtime control.

## Why NuttX + LVGL?

Although NuttX has its own graphic library called [NX](#), LVGL is a good alternative because users could find more eye-candy demos and they can reuse code 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.
  - visualize sensor data directly and easily on the board without using a computer.
  - Final products with a GUI without a touchscreen (i.e. 3D Printer Interface using Rotary Encoder to Input data).
  - Final products with a touchscreen (and all sorts of bells and whistles).
- 

## How to get started with NuttX and LVGL?

There are many boards in the [NuttX mainline](#) with support for LVGL. Let' s use the [STM32F429IDISCOVERY](#) as an example because it is a very popular board.

## First you need to install the pre-requisites on your system

Let' s use the [Windows Subsystem for Linux](#)

```
$ 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 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 `nuttX.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_u
↩halt" -c "flash write_image erase nuttX.bin 0x08000000"
```

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

```
nsh> lvgl_demo
```

### Where can I find more information?

- This blog post: [LVGL on LPCXpresso54628](#)
- NuttX mailing list: [Apache NuttX Mailing List](#)

## 1.2.10 RT-Thread RTOS

### What is RT-Thread?

**RT-Thread** is an [open source](#), neutral, and community-based real-time operating system (RTOS). RT-Thread has **Standard version** and **Nano version**. For resource-constrained microcontroller (MCU) systems, the NANO kernel version that requires only 3 KB Flash and 1.2 KB RAM memory resources can be tailored with easy-to-use tools; And for resource-rich IoT devices, RT-Thread can use the **online software package** management tool, together with system configuration tools, to achieve intuitive and rapid modular cutting, seamlessly import rich software packages, thus achieving complex functions like Android’s graphical interface and touch sliding effects, smart voice interaction effects, and so on.

Key features:

- Designed for resource-constrained devices, the minimum kernel requires only 1.2KB of RAM and 3 KB of Flash.
- A variety of standard interfaces, such as POSIX, CMSIS, C++ application environment.
- Has rich components and a prosperous and fast growing package ecosystem
- Elegant code style, easy to use, read and master.

- High Scalability. RT-Thread has high-quality scalable software architecture, loose coupling, modularity, is easy to tailor and expand.
- Supports high-performance applications.
- Supports all mainstream compiling tools such as GCC, Keil and IAR.
- Supports a wide range of architectures and chips.

## How to run LVGL on RT-Thread?

### 中文文档

LVGL has registered as a [software package](#) of RT-Thread. By using [Env tool](#) or [RT-Thread Studio IDE](#), RT-Thread users can easily download LVGL source code and combine with RT-Thread project. RT-Thread community has port LVGL to several BSPs:

## 1.2.11 CMake

LVGL supports integrating with [CMake](#). It comes with preconfigured targets for:

On top of the preconfigured targets you can also use “plain” CMake to integrate LVGL into any custom C/C++ project.

### Prerequisites

- CMake (  $\geq 3.12.4$  )
- Compatible build tool e.g.

### Building LVGL with CMake

There are many ways to include external CMake projects into your own. A modern one also used in this example is the CMake [FetchContent](#) module. This module conveniently allows us to download dependencies directly at configure time from e.g. [GitHub](#). Here is an example how we might include LVGL into our own project.

```
cmake_minimum_required(VERSION 3.14)
include(FetchContent)

project(MyProject LANGUAGES C CXX)

# Build an executable called "MyFirmware"
add_executable(MyFirmware src/main.c)

# Specify path to own LVGL config header
set(LV_CONF_PATH
    ${CMAKE_CURRENT_SOURCE_DIR}/src/lv_conf.h
    CACHE STRING "" FORCE)

# Fetch LVGL from GitHub
FetchContent_Declare(lvgl URL https://github.com/lvgl/lvgl.git)
FetchContent_MakeAvailable(lvgl)

# The target "MyFirmware" depends on LVGL
target_link_libraries(MyFirmware PRIVATE lvgl::lvgl)
```

This configuration declares a dependency between the two targets **MyFirmware** and **lvgl**. Upon building the target **MyFirmware** this dependency will be resolved and **lvgl** will be built and linked with it. Since LVGL requires a config header called `lv_conf.h` to be includable by its sources we also set the option `LV_CONF_PATH` to point to our own copy of it.

### Additional CMake options

Besides `LV_CONF_PATH` there are two additional CMake options to specify include paths.

`LV_LVGL_H_INCLUDE_SIMPLE` which specifies whether to `#include "lvgl.h"` absolut or relative

`LV_CONF_INCLUDE_SIMPLE` which specifies whether to `#include "lv_conf.h"` and `"lv_drv_conf.h"` absolut or relative

I do not recommend to disable those options unless your folder layout makes it absolutely necessary.

### Building LVGL examples with CMake

LVGL examples have their own CMake target. If you want to build the examples simply add them to your dependencies.

```
# The target "MyFirmware" depends on LVGL and examples
target_link_libraries(MyFirmware PRIVATE lvgl::lvgl lvgl::examples)
```

### Building LVGL drivers and demos with CMake

Exactly the same goes for the `drivers` and the `demos`.

```
# Specify path to own LVGL demos config header
set(LV_DEMO_CONF_PATH
    ${CMAKE_CURRENT_SOURCE_DIR}/src/lv_demo_conf.h
    CACHE STRING "" FORCE)

FetchContent_Declare(lv_drivers
    GIT_REPOSITORY https://github.com/lvgl/lv_drivers)
FetchContent_MakeAvailable(lv_drivers)
FetchContent_Declare(lv_demos
    GIT_REPOSITORY https://github.com/lvgl/lv_demos.git)
FetchContent_MakeAvailable(lv_demos)

# The target "MyFirmware" depends on LVGL, drivers and demos
target_link_libraries(MyFirmware PRIVATE lvgl::lvgl lvgl::drivers lvgl::examples)
```

Just like the `lv_conf.h` header `demos` comes with its own config header called `lv_demo_conf.h`. Analogous to `LV_CONF_PATH` its path can be set by using the option `LV_DEMO_CONF_PATH`.

## 1.3 移植 (Porting)

### 1.3.1 Set up a project

#### Get the library

LVGL 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 itself is the **lvgl** directory which should be copied into your project.

#### Configuration file

There is a configuration header file for LVGL called **lv\_conf.h**. You modify this header to set the library's basic behavior, disable unused modules and features, adjust 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.

Comments in the config file explain the meaning of the options. Be sure to set at least `LV_COLOR_DEPTH` according to your display's color depth.

Alternatively, *lv\_conf.h* can be copied to another place but then you should add the `LV_CONF_INCLUDE_SIMPLE` define to your compiler options (e.g. `-DLV_CONF_INCLUDE_SIMPLE` for GCC compiler) and set the include path manually (e.g. `-I../include/gui`). In this case LVGL will attempt to include *lv\_conf.h* simply with `#include "lv_conf.h"`.

You can even use a different name for *lv\_conf.h*. The custom path can be set via the `LV_CONF_PATH` define. For example `-DLV_CONF_PATH="/home/joe/my_project/my_custom_conf.h"`

If `LV_CONF_SKIP` is defined, LVGL will not try to include *lv\_conf.h*. Instead you can pass the config defines using build options. For example `"-DLV_COLOR_DEPTH=32 -DLV_USE_BTN 1"`. The unset options will get a default value which is the same as the ones in *lv\_conf\_template.h*.

LVGL also can be used via **Kconfig** and **menuconfig**. You can use *lv\_conf.h* together with **Kconfig**, but keep in mind that the value from *lv\_conf.h* or build settings (`-D...`) overwrite the values set in **Kconfig**. To ignore the configs from *lv\_conf.h* simply remove its content, or define `LV_CONF_SKIP`.

#### Initialization

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

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

### 1.3.2 Display interface

To register a display for LVGL, a `lv_disp_draw_buf_t` and a `lv_disp_drv_t` variable have to be initialized.

- `lv_disp_draw_buf_t` contains internal graphic buffer(s) called draw buffer(s).
- `lv_disp_drv_t` contains callback functions to interact with the display and manipulate low level drawing behavior.

#### Draw buffer

Draw buffer(s) are simple array(s) that LVGL uses to render the screen content. Once rendering is ready the content of the draw buffer is sent to the display using the `flush_cb` function set in the display driver (see below).

A draw buffer can be initialized via a `lv_disp_draw_buf_t` variable like this:

```
/*A static or global variable to store the buffers*/
static lv_disp_draw_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). With only one buffer use NULL instead buf_
2 */
lv_disp_draw_buf_init(&disp_buf, buf_1, buf_2, MY_DISP_HOR_RES*10);
```

Note that `lv_disp_draw_buf_t` must be a static, global or dynamically allocated variable. It cannot be a local variable as they are destroyed upon end of scope.

As you can see above, the draw buffer may be smaller than the screen. In this case, larger areas are redrawn in smaller segments that fit into the draw buffer(s). If only a small area changes (e.g. a button is pressed) then only that area will be refreshed.

A larger buffer results in better performance but above 1/10 screen sized buffer(s) there is no significant performance improvement. Therefore it's recommended to choose the size of the draw buffer(s) to be at least 1/10 screen sized.

#### Buffering modes

There are several settings to adjust the number draw buffers and buffering/refreshing modes.

You can measure the performance of different configurations using the [benchmark example](#).

#### One buffer

If only one buffer is used LVGL draws the content of the screen into that draw buffer and sends it to the display. LVGL then needs to wait until the content of the buffer is sent to the display before drawing something new in it.

## Two buffers

If two buffers are used LVGL can draw into one buffer while the content of the other buffer is sent to the display in the background. DMA or other hardware should be used to transfer data to the display so the MCU can continue drawing. This way, the rendering and refreshing of the display become parallel operations.

## Full refresh

In the display driver (`lv_disp_drv_t`) enabling the `full_refresh` bit will force LVGL to always redraw the whole screen. This works in both *one buffer* and *two buffers* modes. If `full_refresh` is enabled and two screen sized draw buffers are provided, LVGL's display handling works like "traditional" double buffering. This means the `flush_cb` callback only has to update the address of the framebuffer (`color_p` parameter). This configuration should be used if the MCU has an LCD controller peripheral and not with an external display controller (e.g. ILI9341 or SSD1963) accessed via serial link. The latter will generally be too slow to maintain high frame rates with full screen redraws.

## Direct mode

If the `direct_mode` flag is enabled in the display driver LVGL will draw directly into a **screen sized frame buffer**. That is the draw buffer(s) needs to be screen sized. In this case `flush_cb` will be called only once when all dirty areas are redrawn. With `direct_mode` the frame buffer always contains the current frame as it should be displayed on the screen. If 2 frame buffers are provided as draw buffers LVGL will alter the buffers but always draw only the dirty areas. Therefore the 2 buffers need to be synchronized in `flush_cb` like this:

1. Display the frame buffer pointed by `color_p`
2. Copy the redrawn areas from `color_p` to the other buffer.

To get the redrawn areas to copy use the following functions `_lv_refr_get_disp_refreshing()` returns the display being refreshed `disp->inv_areas[LV_INV_BUF_SIZE]` contains the invalidated areas `disp->inv_area_joined[LV_INV_BUF_SIZE]` if 1 that area was joined into another one and should be ignored `disp->inv_p` number of valid elements in `inv_areas`

## Display driver

Once the buffer initialization is ready a `lv_disp_drv_t` display driver needs to be:

1. initialized with `lv_disp_drv_init(&disp_drv)`
2. its fields need to be set
3. it needs to be registered in LVGL with `lv_disp_drv_register(&disp_drv)`

Note that `lv_disp_drv_t` also needs to be a static, global or dynamically allocated variable.



## Mandatory fields

In the most simple case only the following fields of `lv_disp_drv_t` need to be set:

- `draw_buf` pointer to an initialized `lv_disp_draw_buf_t` variable.
- `hor_res` horizontal resolution of the display in pixels.
- `ver_res` vertical resolution of the display in pixels.
- `flush_cb` a callback function to copy a buffer's content to a specific area of the display. `lv_disp_flush_ready(&disp_drv)` needs to be called when flushing is ready. LVGL might render the screen in multiple chunks and therefore call `flush_cb` multiple times. To see if the current one is the last chunk of rendering use `lv_disp_flush_is_last(&disp_drv)`.

## Optional fields

There are some optional display driver data fields:

- `physical_hor_res` horizontal resolution of the full / physical display in pixels. Only set this when *not* using the full screen (defaults to -1 / same as `hor_res`).
- `physical_ver_res` vertical resolution of the full / physical display in pixels. Only set this when *not* using the full screen (defaults to -1 / same as `ver_res`).
- `offset_x` horizontal offset from the the full / physical display in pixels. Only set this when *not* using the full screen (defaults to 0).
- `offset_y` vertical offset from the the full / physical display in pixels. Only set this when *not* using the full screen (defaults to 0).
- `color_chroma_key` A color which will be drawn as transparent on chrome keyed images. Set to `LV_COLOR_CHROMA_KEY` from `lv_conf.h` by default.
- `anti_aliasing` use anti-aliasing (edge smoothing). Enabled by default if `LV_COLOR_DEPTH` is set to at least 16 in `lv_conf.h`.
- `rotated` and `sw_rotate` See the [Rotation](#) section below.
- `screen_transp` if 1 the screen itself can have transparency as well. `LV_COLOR_SCREEN_TRANSP` must be enabled in `lv_conf.h` and `LV_COLOR_DEPTH` must be 32.
- `user_data` A custom `void` user data for the driver.
- `full_refresh` always redrawn the whole screen (see above)
- `direct_mode` drive directly into the frame buffer (see above)
- `user_data` A custom `void` user data for the driver..

Some other optional callbacks to make it 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 draw buffer. It can be used to store the pixels more compactly in the draw buffer 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_draw_buf_t` can be smaller to hold only the required number of bits for the given area size. Note that rendering with `set_px_cb` is slower than normal rendering.

- **monitor\_cb** A callback function that tells how many pixels were refreshed and in how much time. Called when the last chunk is rendered and sent to the display.
- **clean\_dcache\_cb** A callback for cleaning any caches related to the display.

LVGL has built-in support to several GPUs (see `lv_conf.h`) but if something else is required these functions can be used to make LVGL use a GPU:

- **gpu\_fill\_cb** fill an area in the memory with a color.
- **gpu\_wait\_cb** if any GPU function returns while the GPU is still working, LVGL will use this function when required to make sure GPU rendering is ready.

## Examples

All together it looks like this:

```
static lv_disp_drv_t disp_drv;           /*A variable to hold the drivers. Must be
↳static or global.*/
lv_disp_drv_init(&disp_drv);              /*Basic initialization*/
disp_drv.draw_buf = &disp_buf;           /*Set an initialized buffer*/
disp_drv.flush_cb = my_flush_cb;         /*Set a flush callback to draw to the
↳display*/
disp_drv.hor_res = 320;                   /*Set the horizontal resolution in pixels*/
disp_drv.ver_res = 240;                   /*Set the vertical resolution in pixels*/

lv_disp_t * disp;
disp = lv_disp_drv_register(&disp_drv); /*Register the driver and save the created
↳display objects*/
```

Here are 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)
↳p)
{
    /*The most simple case (but also the slowest) to put all pixels to the screen one-
    ↳by-one
    *`put_px` is just an example, it needs to be implemented by you.*/
    int32_t x, y;
    for(y = area->y1; y <= area->y2; y++) {
        for(x = area->x1; x <= area->x2; x++) {
            put_px(x, y, *color_p);
            color_p++;
        }
    }

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

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*/
```

(下页继续)

(续上页)

```

    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_rounder_cb(lv_disp_drv_t * disp_drv, lv_area_t * area)
{
    /* Update the areas as needed.
     * For example it makes the area to start only on 8th rows and have Nx8 pixel
     * 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.
     * For example it writes only 1-bit for monochrome displays mapped vertically.*/
    buf += buf_w * (y >> 3) + x;
    if(lv_color_brightness(color) > 128) (*buf) |= (1 << (y % 8));
    else (*buf) &= ~(1 << (y % 8));
}

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

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

```

## Rotation

LVGL supports rotation of the display in 90 degree increments. You can select whether you'd like software rotation or hardware rotation.

If you select software rotation (`sw_rotate` flag set to 1), LVGL will perform the rotation for you. Your driver can and should assume that the screen width and height have not changed. Simply flush pixels to the display as normal. Software rotation requires no additional logic in your `flush_cb` callback.

There is a noticeable amount of overhead to performing rotation in software. Hardware rotation is available to avoid unwanted slow downs. In this mode, LVGL draws into the buffer as if your screen width and height were swapped. You are responsible for rotating the provided pixels yourself.

The default rotation of your display when it is initialized can be set using the `rotated` flag. The available options are `LV_DISP_ROT_NONE`, `LV_DISP_ROT_90`, `LV_DISP_ROT_180`, or `LV_DISP_ROT_270`. The rotation values are relative to how you would rotate the physical display in the clockwise direction. Thus, `LV_DISP_ROT_90` means you rotate the hardware 90 degrees clockwise, and the display rotates 90 degrees counterclockwise to compensate.

(Note for users upgrading from 7.10.0 and older: these new rotation enum values match up with the old 0/1 system for rotating 90 degrees, so legacy code should continue to work as expected. Software rotation is also disabled by default for compatibility.)

Display rotation can also be changed at runtime using the `lv_disp_set_rotation(dis, rot)` API.

Support for software rotation is a new feature, so there may be some glitches/bugs depending on your configuration. If you encounter a problem please open an issue on [GitHub](#).

## Further reading

- [lv\\_port\\_disp\\_template.c](#) for a template for your own driver.
- [Drawing](#) to learn more about how rendering works in LVGL.
- [Display features](#) to learn more about higher level display features.

## API

警告: doxygenfile: Unable to find project ‘lvgl’ in breathe\_projects dictionary

## 1.3.3 Input device interface

### Types of input devices

To register an input device an `lv_indev_drv_t` variable has to be initialized. **Be sure to register at least one display before you register any input devices.**

```
lv_disp_drv_register(&disp_drv);

static 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);
```

The `type` member can be:

- `LV_INDEV_TYPE_POINTER` touchpad or mouse
- `LV_INDEV_TYPE_KEYPAD` keyboard or keypad
- `LV_INDEV_TYPE_ENCODER` encoder with left/right turn and push options
- `LV_INDEV_TYPE_BUTTON` external buttons virtually pressing the screen

`read_cb` is a function pointer which will be called periodically to report the current state of an input device.

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

## Touchpad, mouse or any pointer

Input devices that can click points on the screen belong to this category.

```

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

...

void my_input_read(lv_indev_drv_t * drv, lv_indev_data_t*data)
{
    if(touchpad_pressed) {
        data->point.x = touchpad_x;
        data->point.y = touchpad_y;
        data->state = LV_INDEV_STATE_PRESSED;
    } else {
        data->state = LV_INDEV_STATE_RELEASED;
    }
}

```

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.
- 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;

...

void 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_PRESSED;
    else data->state = LV_INDEV_STATE_RELEASED;
}

```

## Encoder

With an encoder you can do the following:

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 you can navigate inside the object by turning the encoder.
- To leave edit mode, long press 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;

...

void 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_PRESSED;
    else data->state = LV_INDEV_STATE_RELEASED;
}
```

## Using buttons with Encoder logic

In addition to standard encoder behavior, you can also utilize 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 an encoder advance 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;

...
```

(下页继续)

(续上页)

```

void 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_PRESSED;
    else {
        data->state = LV_INDEV_STATE_RELEASED;
        /* Optionally you can also use enc_diff, if you have encoder*/
        data->enc_diff = enc_get_new_moves();
    }
}

```

## 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}, ... }`

**重要:** 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;

...

void 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_PRESSED; /*Set the pressed state*/
    } else {
        data->state = LV_INDEV_STATE_RELEASED; /*Set the released state*/
    }

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

```

## Other features

### Parameters

The default value of the following parameters can be changed in `lv_indev_drv_t`:

- `scroll_limit` Number of pixels to slide before actually scrolling the object.
- `scroll_throw` Scroll throw (momentum) 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_timer` pointer to the `lv_timer` which reads the input device. Its parameters can be changed by `lv_timer_...()` functions. `LV_INDEV_DEF_READ_PERIOD` in `lv_conf.h` sets the default read period.

### Feedback

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 of its type). This allows generating feedback for the user, e.g. to play a sound on `LV_EVENT_CLICKED`.

### Associating with a display

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

### Buffered reading

By default, LVGL calls `read_cb` periodically. Because of this intermittent polling there is a chance that some user gestures are missed.

To solve this you can write an event driven driver for your input device that buffers measured data. In `read_cb` you can report the buffered data instead of directly reading the input device. Setting the `data->continue_reading` flag will tell LVGL there is more data to read and it should call `read_cb` again.

### Further reading

- [lv\\_port\\_indev\\_template.c](#) for a template for your own driver.
- *INdev features* to learn more about higher level input device features.



## API

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## 1.3.4 Tick interface

LVGL needs a system tick to know elapsed time for animations and other tasks.

You need to call the `lv_tick_inc(tick_period)` function periodically and provide the call period in milliseconds. For example, `lv_tick_inc(1)` when calling 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 more time.

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

On Linux based operating systems (e.g. on Raspberry Pi) `lv_tick_inc` can be called in a thread like 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*/
    }
}
```

## API

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## 1.3.5 Task Handler

To handle the tasks of LVGL you need to call `lv_timer_handler()` periodically in one of the following:

- `while(1)` of `main()` function
- timer interrupt periodically (lower priority than `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_timer_handler();
    my_delay_ms(5);
}
```

To learn more about timers visit the [Timer](#) section.

### 1.3.6 Sleep management

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

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

You should also add the following lines to your input device read function to signal a wake-up (press, touch or click etc.) has happened:

```
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 all animations have finished.

### 1.3.7 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\_timer*. Learn more in [Timers](#).

#### 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_timer_handler` and released after it. Also, you have to use the same mutex in other tasks and threads around every LVGL (`lv_...`) related function call and code. 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.

Here is some pseudocode to illustrate the concept:

```
static mutex_t lvgl_mutex;

void lvgl_thread(void)
{
    while(1) {
        mutex_lock(&lvgl_mutex);
        lv_task_handler();
        mutex_unlock(&lvgl_mutex);
    }
}
```

(下页继续)

(续上页)

```

        thread_sleep(10); /* sleep for 10 ms */
    }
}

void other_thread(void)
{
    /* You must always hold the mutex while using LVGL APIs */
    mutex_lock(&lvgl_mutex);
    lv_obj_t *img = lv_img_create(lv_scr_act());
    mutex_unlock(&lvgl_mutex);

    while(1) {
        mutex_lock(&lvgl_mutex);
        /* change to the next image */
        lv_img_set_src(img, next_image);
        mutex_unlock(&lvgl_mutex);
        thread_sleep(2000);
    }
}

```

## Interrupts

Try to avoid calling LVGL functions from interrupt handlers (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_timer_handler` is running.

It's a better approach to simply set a flag or some value in the interrupt, and periodically check it in an LVGL timer (which is run by `lv_timer_handler`).

## 1.3.8 Logging

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

### 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 issues, where the system may fail
- `LV_LOG_LEVEL_USER` Only user messages
- `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.

## Printing logs

### 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`.

### 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(const char * buf)
{
    serial_send(buf, strlen(buf));
}

...

lv_log_register_print_cb(my_log_cb);
```

### Add logs

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

## 1.4 概览 (Overview)

### 1.4.1 对象 (Objects)

在 LVGL 中，用户界面构建的基本单元就是对象，或者叫做 部件 (*Widgets*)。

例如按钮 *Button*，标签 *Label*，图片 *Image*，列表 *List*，图标 *Chart* 或者文本区域 *Text area*。

你可以在对象类型 *Object types* 中查看。

所有对象都使用 “`lv_obj_t`” 指针作为句柄引用。此指针稍后可用于设置或获取对象的属性。

### 属性 (Attributes)

#### 基本属性 (Basic attributes)

所有对象都有如下的基本属性：

- Position 位置
- Size 尺寸
- Parent 形状
- Styles 样式

- Event handlers 事件处理
- Etc 等等

你可以通过函数 `lv_obj_set...` 和 `lv_obj_get...` 来设置/获取属性. 例如:

```
/*Set basic object attributes*/
/* 设置基本属性 */
lv_obj_set_size(btn1, 100, 50);           /*Set a button's size 设置按钮尺寸 */
lv_obj_set_pos(btn1, 20, 30);             /*Set a button's position 设置按钮位置 */
```

要查看所有可用功能, 请访问基础对象文档 *Base object's documentation*.

## 具体属性 (Specific attributes)

对象类型也有特殊的属性。例如, 一个滑块有

- 最大最小值 (Minimum and maximum values )
- 当前值 (Current value)

对于这些特殊的属性, 每个对象类型可能都有唯一的 API 函数。例如对于滑块:

```
/*Set slider specific attributes*/
lv_slider_set_range(slider1, 0, 100);           /*Set
↳ the min. and max. values 设置最大最小值 */
lv_slider_set_value(slider1, 40, LV_ANIM_ON);    /*Set the current value
↳ (position) 设置滑块当前位置 (值) */
```

部件的具体 API 请查看文档 *Documentation* 或者查看他们的头文件 (如 *widgets/lv\_slider.h*)

## 工作机制 (Working mechanisms)

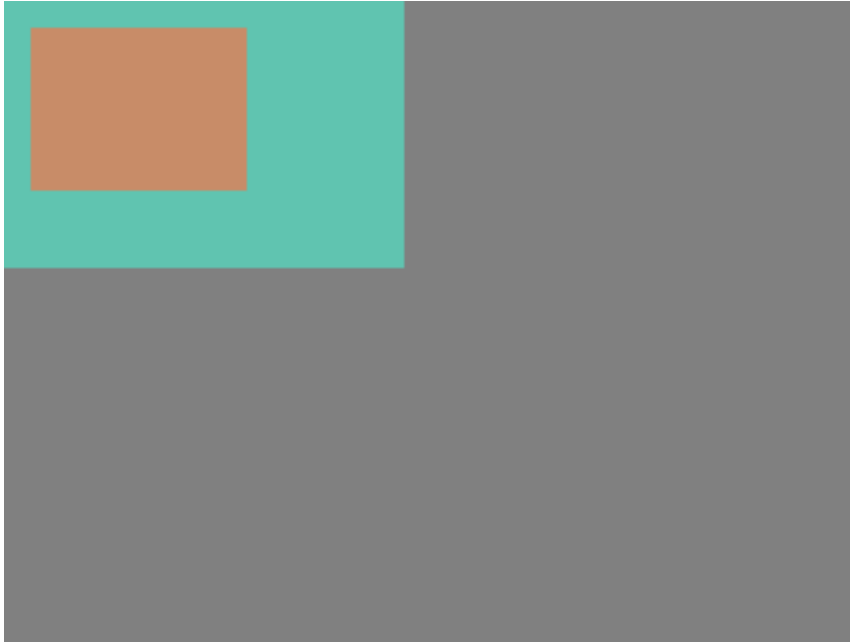
### 父子结构 (Parent-child structure)

父对象可以被视为其子对象的容器。每个对象只有一个父对象 (屏幕除外, 它是顶层对象), 但一个父对象可以有任意数量的子对象。

父对象的类型没有限制, 但有些对象通常是父对象 (例如: 按钮) 或子对象 (例如: 标签)。

### 同时移动 (Moving together)

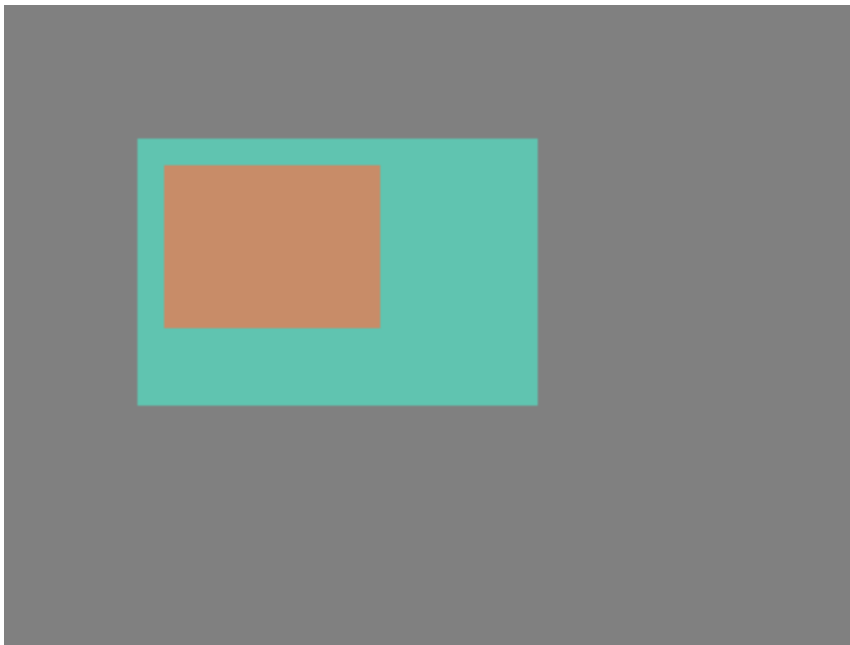
如果父对象的位置发生变化, 子对象将随之移动。因此, 所有位置都相对于父对象而言。



```
lv_obj_t * parent = lv_obj_create(lv_scr_act());    /*Create a parent object on the_
↳current screen 将当前屏幕作为父对象 */
lv_obj_set_size(parent, 100, 80);                  /*Set the size of the_
↳parent 设置父对象尺寸 */

lv_obj_t * obj1 = lv_obj_create(parent);             /*Create an object on the_
↳previously created parent object 为之前的父对象创建一个子对象 */
lv_obj_set_pos(obj1, 10, 10);                       /*Set the position of the_
↳new object 设置子对象位置 */
```

完成父子对象创建后，修改父对象位置：



```
lv_obj_set_pos(parent, 50, 50);           /*Move the parent. The child will move with it.
↪ 子对象同父对象一起移动 */
```

(为简单起见，示例中未显示对象颜色的调整。)

### 子对象的可见性 (Visibility only on the parent)

如果子对象部分或完全位于其父对象之外，则外部部分将不可见。



```
lv_obj_set_x(obj1, -30);                 /*Move the child a little bit off the parent 将子对象移
动一部分出去 */
```

### 创建/删除对象 (Create and delete objects)

在 LVGL 中，可以在运行时动态创建和删除对象。这意味着只有当前创建的（现有）对象消耗 RAM。

这允许仅在单击按钮打开屏幕时创建屏幕，并在加载新屏幕时删除屏幕。

可以根据设备的当前环境创建 UI。例如，可以根据当前连接的传感器创建仪表、图表、条形图和滑块。

每个部件（widgets）的**创建（create）**函数都有如下的原型：

```
lv_obj_t * lv_<widget>_create(lv_obj_t * parent, <other parameters if any>);
```

通常，创建函数只有一个 **parent** 参数，告诉它们在哪个对象上创建新部件。

creat 函数的返回值是一个指向创建对象的指针，类型为 `lv_obj_t *`。

所有对象类型都有一个通用的 **delete** 函数。它删除对象及其所有子对象。

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

- `lv_obj_del` 立即删除对象。

- `lv_obj_del_async(obj)` 异步删除，系统将在下一次调用 `lv_timer_handler()` 时删除对象

例如，如果要在子对象 `LV_EVENT_DELETE` 处理程序中删除此对象的父对象时，可以使用 `lv_obj_clean(obj)` 删除对象的所有子项（但不是对象本身），一段时间后，您可以使用 `lv_obj_del_delayed(obj, 1000)` 来删除对象。延迟以毫秒表示。

## 屏幕 (Screens)

### 创建屏幕 (Create screens)

屏幕是没有父对象的特殊对象，它们可以像这样创建：

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

可以使用任何对象类型创建屏幕对象。具体请看 *Base object*

### 设置活动屏幕 (Get the active screen)

每个显示器上至少需要一个活动屏幕。默认情况下，库创建并加载一个**基础对象**作为屏幕。

要获取当前活动的屏幕，请使用 `lv_scr_act()` 函数。

### 加载屏幕 (Load screens)

要加载新屏幕，请使用 `lv_scr_load(scr1)`。

## 层 (Layers)

系统会自动生成两个层：

- 顶层 top layer
- 系统层 system layer

它们独立于屏幕，并且存在于每个屏幕上。**顶层**在屏幕上的每个对象之上，**系统层**则在**顶层**之上。你可以随意将窗口添加到**顶层**上。但是，**系统层**仅限于系统级的东西（如鼠标光标设置 `lv_indev_set_cursor()`）。

`lv_layer_top()` 和 `lv_layer_sys()` 函数分别返回指向顶层和系统层的指针。

具体请查看 *Layer overview*。

### 用动画加载屏幕 (Load screen with animation)

如果想用动画的方式加载新屏幕，可以使用函数 `lv_scr_load_anim(scr, transition_type, time, delay, auto_del)`。

共有如下几种方式：

- `LV_SCR_LOAD_ANIM_NONE` 在 `delay` 毫秒后立即切换
- `LV_SCR_LOAD_ANIM_OVER_LEFT/RIGHT/TOP/BOTTOM` 将新屏幕按指定的方向移动
- `LV_SCR_LOAD_ANIM_MOVE_LEFT/RIGHT/TOP/BOTTOM` 新屏幕和当前屏幕都按指定的方向移动
- `LV_SCR_LOAD_ANIM_FADE_ON` 将新屏幕淡入旧屏幕



如果将 `auto_del` 设置为 `true`，系统将在动画完成时自动删除旧屏幕。

在动画开始 `delay` 时间后，新屏幕将变为活动状态（由 `lv_scr_act()` 返回）。

### 多显示器处理 (Handling multiple display)

系统在 **当前显示屏 (default display)** 上创建显示，**当前显示屏 (default display)** 由函数 `lv_disp_drv_register` 注册。可以通过函数 `lv_disp_set_default disp` 显式的创建新的 **当前显示屏 (default display)**。

`lv_scr_act()`, `lv_scr_load()` 和 `lv_scr_load_anim()` 都是在 **当前显示屏 (default display)** 上进行的

具体请看 [Multi-display support](#)

### 部件块 (Parts)

部件由多个部件块组成。如 [Base object](#) 使用主控件 (main) 和滚动条 (scrollbar) 控件，[Slider](#) 使用主控件 (main)、指示器 (indicator) 和旋钮 (knob) 部件。

部件块 (Parts) 就像 CSS 中的 *pseudo-elements*

LVGL 中存在以下预定义的部件块 (Parts)：

- `LV_PART_MAIN` 像矩形的背景
- `LV_PART_SCROLLBAR` 滚动条
- `LV_PART_INDICATOR` 指示器，例如用于滑块、条、开关或复选框的勾选框
- `LV_PART_KNOB` 旋钮
- `LV_PART_SELECTED` 指示当前选择的选项或部分
- `LV_PART_ITEMS` 多个相似元素 (如图表中的单元格)
- `LV_PART_TICKS` 刻度，例如在仪表上
- `LV_PART_CURSOR` 光标，标记特定区域
- `LV_PART_CUSTOM_FIRST` 自定义

部件块的主要目的是允许为部件的“组件”设置样式，具体请看 [Style overview](#) .

### 状态 (States)

对象可以有以下一种或多种状态：

- `LV_STATE_DEFAULT` 正常、释放状态
- `LV_STATE_CHECKED` 切换或选中状态
- `LV_STATE_FOCUSED` 被键盘、编码器或触摸板/鼠标选中
- `LV_STATE_FOCUS_KEY` 通过键盘或编码器选中，但不通过触摸板/鼠标选中
- `LV_STATE_EDITED` 由编码器编辑
- `LV_STATE_HOVERED` 鼠标悬停（暂不支持）
- `LV_STATE_PRESSED` 点击中
- `LV_STATE_SCROLLLED` 滚动中

- LV\_STATE\_DISABLED 去使能状态
- LV\_STATE\_USER\_1 用户定义
- LV\_STATE\_USER\_2 用户定义
- LV\_STATE\_USER\_3 用户定义
- LV\_STATE\_USER\_4 用户定义

当用户与对象交互（按下、释放、选中等）时，LVGL 通常会自动更改状态，当然也可以手动更改。

设置或清除给定状态（但保持其他状态不变）使用 `lv_obj_add/clear_state(obj, LV_STATE_...)`

可以使用或运算符同时赋值两个状态 `lv_obj_add_state(obj, part, LV_STATE_PRESSED | LV_STATE_CHECKED)`。

具体请参阅[Style overview](#)。

## 截屏 (Snapshot)

可以为对象及其子对象生成截屏图像。具体请参阅[Snapshot](#)。

## 1.4.2 位置、大小与布局 (Positions, sizes, and layouts)

### 概览 (Overview)

与 LVGL 的许多其他部分类似，设置坐标的概念受到 CSS 的启发。

简单来说：

- 坐标显式地存储在样式（大小、位置、布局等）中
- 支持最小宽度、最大宽度、最小高度、最大高度
- 有像素、百分比和“内容”单位
- `x=0; y=0` 坐标表示父对象的左上角加左/上边距加上边框宽度
- 宽度/高度表示完整尺寸，“内容区域”较小于填充和边框宽度
- 支持部分 flexbox 和 grid 功能

### 单位 (Units)

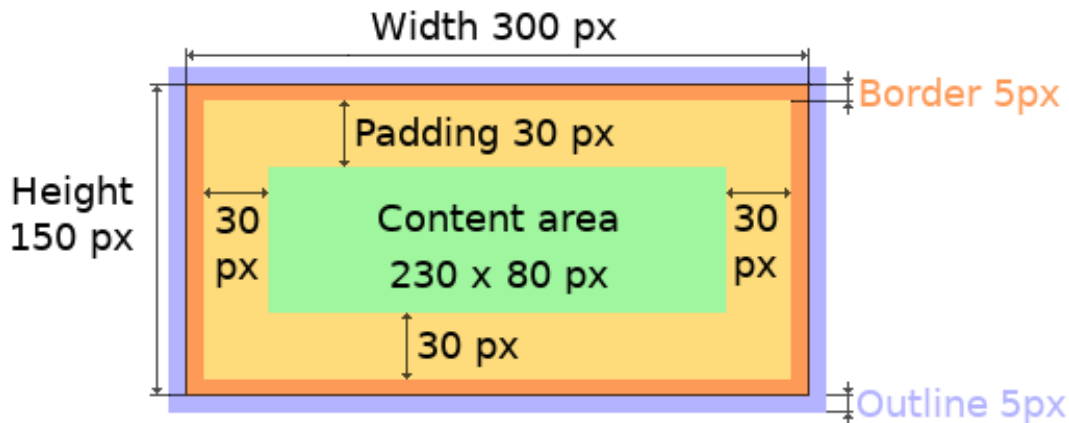
- 像素 pixel: 像素坐标，只能为整数. E.g. `lv_obj_set_x(btn, 10)`
- 百分比 percentage: 对象或其父对象的大小百分比（取决于属性）。`lv_pct(value)` 将值转换为百分比。E.g. `lv_obj_set_width(btn, lv_pct(50))`
- LV\_SIZE\_CONTENT: 设置对象的宽度/高度以涉及所有子项的特殊值。就像 CSS 中的 `auto`。E.g. `lv_obj_set_width(btn, LV_SIZE_CONTENT)`。

## 盒子模型 (Boxing model)

与 CSS 中的模型相似 `border-box` .

对象的 “box” 由以下部分构成：

- 边界框 (bounding box)：元素的宽度/高度。
- 边框宽度 (border width)：边框的宽度。
- 填充 (padding)：对象两侧与其子对象之间的空间。
- 内容 (content)：内容区域，即边界框的大小减去边框宽度和内边距。



边框绘制在边界框内，放置对象的子对象时，在边界内 LVGL 会保留 “填充边距”。

轮廓绘制在边界框外。

## 重要提示 (Important notes)

本节描述了 LVGL 可能会出现的情况。

## 非立即计算坐标 (Postponed coordinate calculation)

LVGL 为了提高性能，不会立即重新计算坐标变化，变化的对象被标记为 “dirty”，在重绘屏幕之前 LVGL 会检查是否有 “dirty” 对象，如果有，则刷新它们的位置、大小和布局。

换句话说，如果你需要获取一个物体的坐标，而坐标刚刚改变，则需要强制 LVGL 重新计算坐标。

具体的，使用函数 `lv_obj_update_layout(obj)`。

对象的大小和位置可能取决于父级或布局，调用 `lv_obj_update_layout` 函数会重新计算 `obj` 屏幕上所有对象的坐标。

## 移除样式 (Removing styles)

正如使用样式 *Using styles* 部分所述，坐标也可以通过样式属性设置。更准确地说，每个与样式坐标相关的属性都存储为样式属性。如果你使用 `lv_obj_set_x(obj, 20)` 函数，LVGL 会将 `x=20` 存储在对象的样式中。

这是一个内部机制，与使用 LVGL 没有多大关系。但是，有一种情况需要了解。如果对象的样式被删除

```
lv_obj_remove_style_all(obj)
```

或者

```
lv_obj_remove_style(obj, NULL, LV_PART_MAIN);
```

先前设置的坐标也将被删除。

例如：

```
/*obj1 的大小最终将被设置为默认值 */
lv_obj_set_size(obj1, 200, 100); /*Now obj1 has 200;100 size*/
lv_obj_remove_style_all(obj1); /*It removes the set sizes*/

/*obj2 会被设置为尺寸 200, 100 */
lv_obj_remove_style_all(obj2);
lv_obj_set_size(obj2, 200, 100);
```

## 坐标 (Position)

### 简易方式 (Simple way)

要设置对象的 x 和 y 坐标，请使用：

```
lv_obj_set_x(obj, 10); //分开设置
lv_obj_set_y(obj, 20);
lv_obj_set_pos(obj, 10, 20); //同时设置
```

默认情况下，x 和 y 坐标是从父级内容区域的左上角开始测量的。

比如，如果父对象设置了每边 5 个像素的填充，上述代码实际上会在 (15, 25) 的位置放置 `obj`。

百分比值则是根据父级的内容区域大小计算的。

```
lv_obj_set_x(btn, lv_pct(10)); //x = 10 % of parent content area width
```

## 对齐 (Align)

在某些情况下，可以从默认左上角更改定位原点。例如更改为右下角，(0,0) 位置表示：与右下角对齐。

使用函数：

```
lv_obj_set_align(obj, align);
```

如果要更改对齐方式并设置新坐标，请执行以下操作：

```
lv_obj_align(obj, align, x, y);
```

可以使用以下对齐选项：

- LV\_ALIGN\_TOP\_LEFT
- LV\_ALIGN\_TOP\_MID
- LV\_ALIGN\_TOP\_RIGHT
- LV\_ALIGN\_BOTTOM\_LEFT
- LV\_ALIGN\_BOTTOM\_MID
- LV\_ALIGN\_BOTTOM\_RIGHT
- LV\_ALIGN\_LEFT\_MID
- LV\_ALIGN\_RIGHT\_MID
- LV\_ALIGN\_CENTER

由于将子对象与其父对象的中心对齐是常用方式，因此存在一个专用函数：

```
lv_obj_center(obj);  
  
//同样效果  
lv_obj_align(obj, LV_ALIGN_CENTER, 0, 0);
```

如果父对象的大小更改，则子对象的设置对齐方式和位置将自动更新。

上面介绍的函数将对象与其父对象对齐。但也可以将对象与任意引用对象对齐。

```
lv_obj_align_to(obj_to_align, reference_obj, align, x, y);
```

除了上面的对齐选项外，还可以使用以下选项对齐外部的对象：

- LV\_ALIGN\_OUT\_TOP\_LEFT
- LV\_ALIGN\_OUT\_TOP\_MID
- LV\_ALIGN\_OUT\_TOP\_RIGHT
- LV\_ALIGN\_OUT\_BOTTOM\_LEFT
- LV\_ALIGN\_OUT\_BOTTOM\_MID
- LV\_ALIGN\_OUT\_BOTTOM\_RIGHT
- LV\_ALIGN\_OUT\_LEFT\_TOP
- LV\_ALIGN\_OUT\_LEFT\_MID
- LV\_ALIGN\_OUT\_LEFT\_BOTTOM
- LV\_ALIGN\_OUT\_RIGHT\_TOP
- LV\_ALIGN\_OUT\_RIGHT\_MID
- LV\_ALIGN\_OUT\_RIGHT\_BOTTOM

例如，要对齐按钮上方的标签并使标签水平居中，请执行以下操作：

```
lv_obj_align_to(label, btn, LV_ALIGN_OUT_TOP_MID, 0, -10);
```

请注意，与 `lv_obj_align()` 不同，`lv_obj_align_to()` 不能在对象坐标或参考对象坐标发生变化时重新对齐对象。

## 尺寸 (Size)

### 简易方式 (Simple way)

同样的，可以通过如下方式设置尺寸

```
lv_obj_set_width(obj, 200);           //Separate...
lv_obj_set_height(obj, 100);
lv_obj_set_size(obj, 200, 100);      //Or in one function
```

百分比值是基于父级内容区域大小计算的。例如，要将对象的高度设置为屏幕高度：

```
lv_obj_set_height(obj, lv_pct(100));
```

尺寸大小设置支持一个特殊值: `LV_SIZE_CONTENT`. 这意味着对象在各方向上的大小将设置为其子对象的大小。

请注意，此时只会考虑右侧和底部的子项，而顶部和左侧的子项仍会被裁剪，同时，带有 `LV_OBJ_FLAG_HIDDEN` 或 `LV_OBJ_FLAG_FLOATING` 的对象将被“`LV_SIZE_CONTENT`”计算忽略。

上述函数设置对象边界框的大小，同样的我们也可以设置内容区域的大小。这意味着对象的边界框会因为填充的增加而扩大。

```
lv_obj_set_content_width(obj, 50); //实际宽度: padding left + 50 + padding right
lv_obj_set_content_height(obj, 30); //实际宽度: padding top + 30 + padding bottom
```

可以使用以下函数检索边界框和内容区域的大小：

```
lv_coord_t w = lv_obj_get_width(obj);
lv_coord_t h = lv_obj_get_height(obj);
lv_coord_t content_w = lv_obj_get_content_width(obj);
lv_coord_t content_h = lv_obj_get_content_height(obj);
```

### 使用样式 (Using styles)

在底层中，位置、大小和对齐特性实际上是样式特性。为了简单起见，上述“简单方式”没有使用与样式相关的代码，而是在对象的本地样式中设置位置、大小和对齐特性。

但是，使用样式设置坐标有一定优势：

- 它可以方便地同时设置多个对象的宽度/高度等。例如，使所有滑块的大小为 100x10 像素。
- 它可以在一个位置修改值。
- 这些值可以被其他样式部分覆盖。例如，`style_btn` 默认设置对象为“100x50”，但添加 `style_full_width` 只覆盖对象的宽度。
- 根据状态，对象可以具有不同的位置或大小，如在 `LV_STATE_DEFAULT` 状态下宽度为 100 px，在 `LV_STATE_PRESSED` 状态下宽度为 120 px
- 样式转换使坐标更改更加平滑。

具体的：

```
static lv_style_t style;
lv_style_init(&style);
lv_style_set_width(&style, 100);
```

(下页继续)

(续上页)

```
lv_obj_t * btn = lv_btn_create(lv_scr_act());
lv_obj_add_style(btn, &style, LV_PART_MAIN);
```

正如您将在下面看到的，LVGL 还有其他一些关于大小和位置设置的强大功能。但是，为了保持 LVGL API 的精简，最常见是使用简单方式的坐标设置功能，更复杂的功能可以通过样式方法来使用。

### 位置变换 (Translation)

假设有 3 个按钮彼此相邻。其位置设置如上边的代码所述，如何实现，当按下按钮时，按钮向上移动一点？实现这一点的一种方法是为**按下状态**设置新的 Y 坐标：

```
static lv_style_t style_normal;
lv_style_init(&style_normal);
lv_style_set_y(&style_normal, 100);

static lv_style_t style_pressed;
lv_style_init(&style_pressed);
lv_style_set_y(&style_pressed, 80);

lv_obj_add_style(btn1, &style_normal, LV_STATE_DEFAULT);
lv_obj_add_style(btn1, &style_pressed, LV_STATE_PRESSED);

lv_obj_add_style(btn2, &style_normal, LV_STATE_DEFAULT);
lv_obj_add_style(btn2, &style_pressed, LV_STATE_PRESSED);

lv_obj_add_style(btn3, &style_normal, LV_STATE_DEFAULT);
lv_obj_add_style(btn3, &style_pressed, LV_STATE_PRESSED);
```

这是可行的，但它不够灵活，因为按下的坐标是硬编码的。如果按钮不在 y=100，`style_pressed` 无法正常工作，位置变换可以解决这个问题。

```
static lv_style_t style_normal;
lv_style_init(&style_normal);
lv_style_set_y(&style_normal, 100);

static lv_style_t style_pressed;
lv_style_init(&style_pressed);
lv_style_set_translate_y(&style_pressed, -20); //Translation

lv_obj_add_style(btn1, &style_normal, LV_STATE_DEFAULT);
lv_obj_add_style(btn1, &style_pressed, LV_STATE_PRESSED);

lv_obj_add_style(btn2, &style_normal, LV_STATE_DEFAULT);
lv_obj_add_style(btn2, &style_pressed, LV_STATE_PRESSED);

lv_obj_add_style(btn3, &style_normal, LV_STATE_DEFAULT);
lv_obj_add_style(btn3, &style_pressed, LV_STATE_PRESSED);
```

位置变换是从对象的当前位置开始的，所以和初始坐标无关

百分比值也可以用于位置变换。百分比是相对于对象本身的大小而言（而不是相对于父对象的大小）。如 `lv_pct(50)` 将以对象本身的宽度/高度的一半移动对象。

由于是在计算布局之后进行变换的，因此，可以平移已有对象的位置。

变换实际上移动了对象，这意味着它会使滚动条和 LV\_SIZE\_CONTENT 大小的对象对位置变化做出反应。

### 变形 (Transformation)

与位置类似，对象的大小也可以相对于当前大小进行更改。转换后的宽度和高度将添加到对象的两侧。这意味着 10 px 变换宽度会使对象宽 2x10 像素。

与位置平移不同的是，尺寸变换不会使对象“真正”变大。换句话说，滚动条、布局 LV\_SIZE\_CONTENT 不会对转换后的大小做出反应。

因此，变换“仅”是一种视觉效果。

此代码在按下时放大按钮：

```
static lv_style_t style_pressed;
lv_style_init(&style_pressed);
lv_style_set_transform_width(&style_pressed, 10);
lv_style_set_transform_height(&style_pressed, 10);

lv_obj_add_style(btn, &style_pressed, LV_STATE_PRESSED);
```

### 最大最小值 (Min and Max size)

与 CSS 类似，LVGL 也支持 min-width、max-width、min-height 和 max-height。这是为了防止对象大小变得比这些值更小/更大。

如果大小是按百分比或“LV\_SIZE\_CONTENT”设置的话，这个设置会更加有用。

```
static lv_style_t style_max_height;
lv_style_init(&style_max_height);
lv_style_set_y(&style_max_height, 200);

lv_obj_set_height(obj, lv_pct(100));
lv_obj_add_style(obj, &style_max_height, LV_STATE_DEFAULT); //Limit the height to
↪200 px
```

也可以使用相对于父内容区域大小的百分比值。

```
static lv_style_t style_max_height;
lv_style_init(&style_max_height);
lv_style_set_y(&style_max_height, lv_pct(50));

lv_obj_set_height(obj, lv_pct(100));
lv_obj_add_style(obj, &style_max_height, LV_STATE_DEFAULT); //Limit the height to
↪half parent height
```



## 布局 (Layout)

### 概览

布局可以更新对象子项的位置和大小。它们可用于自动将子项排列成一行或一列，或者以更复杂的形式排列。

布局设置的位置和大小会覆盖“正常”的 x、y、宽度和高度设置。

每个布局只有一个相同的功能：`lv_obj_set_layout(obj, <LAYOUT_NAME>)` 在对象上设置布局。

有关父级和子级的更多设置，请参阅给定布局的文档。

### 内置布局 (Built-in layout)

LVGL 带有两个非常强大的布局：

- Flexbox
- Grid

### 标志 (Flags)

有一些标志可用于对象以影响它们在布局中的行为：

- `LV_OBJ_FLAG_HIDDEN` 在布局计算中忽略隐藏的对象。
- `LV_OBJ_FLAG_IGNORE_LAYOUT` 该对象会被布局简单地忽略。它的坐标可以照常设置。
- `LV_OBJ_FLAG_FLOATING` 与 `LV_OBJ_FLAG_IGNORE_LAYOUT` 相同，但在 `LV_SIZE_CONTENT` 计算中将忽略带有 `LV_OBJ_FLAG_FLOATING` 的对象。

这些标志可以添加/删除 `lv_obj_add/clear_flag(obj, FLAG);`

### 添加新布局 (Adding new layouts)

LVGL 可以自定义布局自由扩展：

```
uint32_t MY_LAYOUT;

...

MY_LAYOUT = lv_layout_register(my_layout_update, &user_data);

...

void my_layout_update(lv_obj_t * obj, void * user_data)
{
    /*Will be called automatically if it's required to reposition/resize the
    ↪ children of "obj" */
}
```

可以添加自定义样式属性，这些属性可以在更新回调中检索和使用。例如：

```

uint32_t MY_PROP;
...

LV_STYLE_MY_PROP = lv_style_register_prop();

...
static inline void lv_style_set_my_prop(lv_style_t * style, uint32_t value)
{
    lv_style_value_t v = {
        .num = (int32_t)value
    };
    lv_style_set_prop(style, LV_STYLE_MY_PROP, v);
}

```

## Examples

### 1.4.3 样式 (Styles)

*Styles* 用于设置对象的外观。lvgl 中的样式很大程度上受到 CSS 的启发。简而言之，其概念如下：

- 样式是一个 `lv_style_t` 变量，它可以保存边框宽度、文本颜色等属性。它类似于 CSS 中的“类”。
- 可以将样式分配给对象以更改其外观。在赋值时，可以指定目标部分（CSS 中的 *pseudo-element*）和目标状态（*pseudo class*）。例如，当滑块处于按下状态时，可以将“`style_blue`”样式添加到滑块的旋钮。
- 任何数量的对象都可以使用相同的样式。
- 样式可以级联，这意味着可以将多个样式分配给同一个对象，并且每个样式可以具有不同的属性。因此，并非所有属性都必须在样式中指定。LVGL 将搜索一个属性，直到一个样式定义它，或者如果它没有被任何样式指定，则使用默认值。例如，`style_btn` 可以导致默认的灰色按钮，而 `style_btn_red` 只能添加一个 `background-color=red` 来覆盖背景颜色。
- 最近添加的样式具有更高的优先级。这意味着如果一个属性以两种样式指定，则将使用对象中的最新样式。
- 如果未在对象中指定，某些属性（例如文本颜色）可以从父级继承。
- 对象也可以有比“正常”样式更高优先级的本地样式。
- 与 CSS（伪类描述不同的状态，例如：`:focus`）不同，在 LVGL 中，一个属性被分配给给定的状态。
- 当对象改变状态时可以使用转换或者变换。

### 状态 (States)

对象可以处于以下状态的组合：

- `LV_STATE_DEFAULT` 正常、释放状态
- `LV_STATE_CHECKED` 切换或选中状态
- `LV_STATE_FOCUSED` 被键盘、编码器或触摸板/鼠标选中
- `LV_STATE_FOCUS_KEY` 通过键盘或编码器选中，但不通过触摸板/鼠标选中
- `LV_STATE_EDITED` 由编码器编辑
- `LV_STATE_HOVERED` 鼠标悬停（暂不支持）
- `LV_STATE_PRESSED` 点击中

- LV\_STATE\_SCROLLLED 滚动中
- LV\_STATE\_DISABLED 去使能状态
- LV\_STATE\_USER\_1 用户定义
- LV\_STATE\_USER\_2 用户定义
- LV\_STATE\_USER\_3 用户定义
- LV\_STATE\_USER\_4 用户定义

一个对象可以同时处于聚焦和按下等状态的组合。这表示为 LV\_STATE\_FOCUSED | LV\_STATE\_PRESSED.

可以将样式添加到任何状态或状态组合。例如，为默认状态和按下状态设置不同的背景颜色。如果属性未在状态中定义，则将使用最佳匹配状态的属性。通常这意味着使用属性 LV\_STATE\_DEFAULT。如果即使为默认状态也未设置该属性，则将使用默认值。（见后）

但**最佳匹配属性**到底意味着什么？状态有一个由其值显示的优先级（请参见上面的列表）。值越大，优先级越高。为了确定要使用哪个状态的属性，让我们举一个例子。假设背景色的定义如下：

- LV\_STATE\_DEFAULT: white
- LV\_STATE\_PRESSED: gray
- LV\_STATE\_FOCUSED: red

1. 最初，对象处于默认状态，因此情况很简单：属性在对象的当前状态中完全定义为白色。
2. 按下对象时，有两个相关属性：默认为白色（默认为与每个状态相关）和按下为灰色。按下状态具有 0x0020 优先级，高于默认状态的 0x0000 优先级，因此将使用灰色。
3. 当对象被聚焦时，与按下状态下发生的情况相同，将使用红色。（选中状态的优先级高于默认状态）。
4. 当对象聚焦并按下时，灰色和红色都可以工作，但按下状态的优先级高于聚焦状态，因此将使用灰色。
5. 但也可以为设置为玫瑰色 LV\_STATE\_PRESSED | LV\_STATE\_FOCUSED. 在这种情况下，此组合状态的优先级为 0x0020+0x0002=0x0022，高于按下状态的优先级，因此将使用玫瑰色（红加灰）。
6. 当对象处于选中状态时，没有属性可设置此状态的背景色。因此，由于没有更好的选项，对象在默认状态的属性中保持为白色。

一些实用说明：

- 状态的优先级（值）非常直观，这是用户期望的。例如，如果一个对象被聚焦，用户仍然希望看到它是否被按下，因此按下状态具有更高的优先级。如果聚焦状态具有更高的优先级，它将覆盖按下的颜色。
- 如果要为所有状态设置属性（例如，红色背景色），只需将其设置为默认状态。如果对象找不到其当前状态的属性，它将返回默认状态的属性。
- 使用 OR 运算符描述复杂情况的属性。（例如，按下 + 选中 + 聚焦）
- 为不同的状态使用不同的样式元素是个好习惯。例如，想要找到按下、选中 + 按下、聚焦、聚焦 + 按下、聚焦 + 按下、聚焦 + 按下 + 选中等状态时的背景色相当困难。相反，例如，对按下和选中的状态使用背景色，并用不同的边框颜色指示聚焦状态可能会更好。
-

## 层叠样式 (Cascading styles)

不需要在一种样式中设置所有属性。可以向对象添加更多样式，并使后来添加的样式修改或扩展外观。例如，创建常规灰色按钮样式，并为仅设置了新背景色的红色按钮创建新样式。

这与 CSS 中列出的类非常相似 `<div class=".btn .btn-red">`.

以后添加的样式优先于以前设置的样式。因此，在上面的灰色/红色按钮示例中，应首先添加普通按钮样式，然后添加红色样式。但是，仍然需要考虑到状态的优先级，比如：

- 基本按钮样式为默认状态定义深灰色，为按下状态定义浅灰色
- 红色按钮样式在默认状态下将背景颜色定义为红色

在这种情况下，当按钮被释放（处于默认状态）时，它将是红色的，因为在最近添加的样式（红色）中找到了完美匹配。当按钮被按下时，浅灰色是更好的匹配，因为它完美地描述了当前状态，所以按钮将是浅灰色。

## 继承 (Inheritance)

某些属性（通常与文本相关的属性）可以从父对象的样式继承。仅当未在对象的样式中设置给定属性时（即使在默认状态下），才应用继承。

在这种情况下，如果该属性是可继承的，则将在父对象中搜索该属性的值，直到某个对象为该属性指定了一个值。父对象将使用自己的状态来确定值。因此，如果按下按钮，并且文本颜色来自此处，则将使用按下的文本颜色。

## 部件块 (Parts)

对象由**部件块**组成，每个部件块都有自己的样式。

LVGL 中存在以下预定义的部件块：

- LV\_PART\_MAIN 像矩形的背景
- LV\_PART\_SCROLLBAR 滚动条
- LV\_PART\_INDICATOR 指示器，例如用于滑块、条、开关或复选框的勾选框
- LV\_PART\_KNOB 旋钮
- LV\_PART\_SELECTED 指示当前选择的选项或部分
- LV\_PART\_ITEMS 多个相似元素 (如图表中的单元格)
- LV\_PART\_TICKS 刻度，例如在仪表上
- LV\_PART\_CURSOR 光标，标记特定区域
- LV\_PART\_CUSTOM\_FIRST 自定义

如一个滑块 *Slider* 有以下几个部件块：

- 背景
- 指示器
- 旋钮

这意味着滑块的三个部分都可以有自己的样式，请参见下文如何向对象和部件块添加样式。

## 初始化样式和设置/读取属性 (Initialize styles and set/get properties)

样式存储在 `lv_style_t` 变量中。样式变量应为静态 `static`、全局 `global` 或动态分配 `dynamically allocated`。

换句话说，它们不能是函数中的局部变量，因为函数退出时局部变量会被销毁。

在使用样式之前，应使用 `lv_style_init(&my_style)` 对其进行初始化。初始化样式后，可以添加或更改特性。

使用函数 `lv_style_set_<property_name>(&style, <value>)`；来设置样式，例如：

```
static lv_style_t style_btn; //定义样式变量
lv_style_init(&style_btn); //初始化样式变量
lv_style_set_bg_color(&style_btn, lv_color_hex(0x115588)); //设置颜色属性
lv_style_set_bg_opa(&style_btn, LV_OPA_50);
lv_style_set_border_width(&style_btn, 2);
lv_style_set_border_color(&style_btn, lv_color_black());

static lv_style_t style_btn_red;
lv_style_init(&style_btn_red);
lv_style_set_bg_color(&style_btn_red, lv_palette_main(LV_PALETTE_RED));
lv_style_set_bg_opa(&style_btn_red, LV_OPA_COVER);
```

如果想要删除属性可以使用如下的方式：

```
lv_style_remove_prop(&style, LV_STYLE_BG_COLOR);
```

如果想要读取属性可以使用如下的方式：

```
lv_style_value_t v;
lv_res_t res = lv_style_get_prop(&style, LV_STYLE_BG_COLOR, &v);
if(res == LV_RES_OK) { /*Found*/
    do_something(v.color);
}
```

`lv_style_value_t` 有 3 个字段：

- `num` 是整数、布尔值和不透明度属性
- `color` 是颜色属性
- `ptr` 是指针属性

要重置样式（清除其所有数据），请使用：

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

样式也可以构建为 `const` 以保存在 RAM 中：

```
const lv_style_const_prop_t style1_props[] = {
    LV_STYLE_CONST_WIDTH(50),
    LV_STYLE_CONST_HEIGHT(50),
    LV_STYLE_PROP_INV,
};

LV_STYLE_CONST_INIT(style1, style1_props);
```

`const` 定义的样式可以像其他样式一样使用，但（显然）不能添加新属性。

## 向部件添加和删除样式 (Add and remove styles to a widget)

如果仅仅是定义一种样式并没有什么用，我们需要把样式指定给部件（widget）才能发挥它的效果。

### 添加样式 (Add styles)

使用函数 `lv_obj_add_style(obj, &style, <selector>)` 向一个对象（object）添加样式。`<selector>` 是一个可以通过或运算定义的参数，它代表向某个部件块的某个状态添加样式，例如：

- `LV_PART_MAIN | LV_STATE_DEFAULT`
- `LV_STATE_PRESSED`: 代表主部件被按下时 `LV_PART_MAIN` 可以省略
- `LV_PART_SCROLLBAR`: 代表默认状态时的滚动条
- `LV_STATE_DEFAULT` 可以省略
- `LV_PART_SCROLLBAR | LV_STATE_SCROLLLED`: 代表对象滚动时的滚动条部分
- `0` 和 `LV_PART_MAIN | LV_STATE_DEFAULT` 的含义一样
- `LV_PART_INDICATOR | LV_STATE_PRESSED | LV_STATE_CHECKED` 代表指示器被按下并被选中时

使用样式 `lv_obj_add_style`:

```
lv_obj_add_style(btn, &style_btn, 0); /* 按钮默认
样式 Default button style*/
lv_obj_add_style(btn, &btn_red, LV_STATE_PRESSED); /* 按钮按下时变为红色 Overwrite only,
↪ some colors to red when pressed*/
```

### 删除样式 (Remove styles)

使用函数 `lv_obj_remove_style_all(obj)` 来删除对象的所有样式。

使用函数 `lv_obj_remove_style(obj, style, selector)` 来删除指定部件块、状态的样式。此函数只会在匹配上在执行函数 `lv_obj_add_style` 时设置的 `selector` 参数之后才执行删除操作。

参数 `style` 可以为 `NULL` 来达到删除 `selector` 里的所有样式

参数 `selector` 可以使用 `LV_STATE_ANY` 和 `LV_PART_ANY` 两个值来删除任何状态或者任何部件块的样式

### 报告样式变更 (Report style changes)

如果已分配给对象的样式发生更改（即添加或更改属性），则应通知使用该样式的对象。有 3 个选项可以执行此操作：

1. 如果更改的属性可以通过简单的重绘（例如更改颜色或不透明度）来更新，只需调用函数 `lv_obj_invalidate(obj)` 或 `lv_obj_invalidate(lv_scr_act())`。
2. 如果更改或添加了更复杂的样式属性，并且知道哪些对象调用了该样式，则调用函数 `lv_obj_refresh_style(obj, part, property)`。要刷新所有部件块和属性，请使用 `lv_obj_refresh_style(obj, LV_PART_ANY, LV_STYLE_PROP_ANY)`。
3. 如果想要检查某个对象的样式是否被更改并同时刷新它们，可以调用函数 `lv_obj_report_style_change(&style)` 来实现。如果 `style` 为 `NULL` 则所有对象都会收到样式被更改的报告

## 获取某个对象的属性值 (Get a property's value on an object)

由于对象的样式存在级联、继承、局部样式和变换，某些时候我们可以需要获得属性的最终值，可以通过函数 `lv_obj_get_style_<property_name>(obj, <part>)` 来实现。

函数会读取对象的当前状态，如果当前状态的属性值未被更改，则返回默认值。

例如：

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

## 局部样式 (Local styles)

除了普通样式之外，对象还可以存储局部样式。这个概念类似于 CSS 中做了修改的内联样式 (e.g. `<div style="color:red">`)。

局部样式与普通样式类似，但它们不能在其他对象之间共享。如果使用，局部样式会自动分配，并在删除对象时释放。在向对象添加局部定义时很有用。

与 CSS 不同，LVGL 局部样式可以分配给状态 (*pseudo-classes*) 和部分 (*pseudo-elements*)。

使用函数 `lv_obj_set_style_<property_name>(obj, <value>, <selector>)` 来分配局部样式

例如：

```
lv_obj_set_style_bg_color(slider, lv_color_red(), LV_PART_INDICATOR | LV_STATE_
↪ FOCUSED);
```

## 属性 (Properties)

请看[here](#).

## 典型背景属性 (Typical background properties)

在部件块的文档中，您将看到“部件块使用典型的背景属性”这样的句子。这些“典型背景属性”与以下相关：

- 背景 Background
- 边界 Border
- 轮廓 Outline
- 影子 Shadow
- 填充 Padding
- 宽度和高度变换 Width and height transformation
- X 和 Y 变换 X and Y translation

## 变换 (Transitions)

默认情况下，当对象更改状态时（例如按下），对象将立即更新到新状态所对应的属性。如果使用变换的方式，我们可以实现状态更改的动画效果。

例如，按下按钮时，其背景颜色可以在 300 毫秒内变化为按下状态颜色。

变换的参数同样存储在样式里并且可以设置：

- 变换时间 the time of the transition
- 延迟变换时间 the delay before starting the transition
- 动画路径（也称为计时或缓动功能） the animation path (also known as the timing or easing function)
- 动画的属性 the properties to animate

可以为每个状态定义变换属性，例如，给默认状态设置 500ms 的变换过程，当对象变为默认状态时就会使用 500ms 变换这个参数，同时，给按下状态设置 100 ms 的转换时间，则进入按下状态时，经历 100ms 的变换过程。

此这个例子的效果就是按钮快速进入按下状态，然后缓慢返回默认状态。

如果我们需要变换这一功能的话，则需要初始化变量 `lv_transition_dsc_t`。

```
/*Only its pointer is saved so must static, global or dynamically allocated */
/* 由于传进去是指针，所以必须因为静态、全局或动态分配 */
static const lv_style_prop_t trans_props[] = {
    ↪ STYLE_BG_OPA, LV_STYLE_BG_COLOR,
    ↪ /*End marker*/
};

static lv_style_transition_dsc_t trans1;
lv_style_transition_dsc_init(&trans1, trans_props, lv_anim_path_ease_out, duration_ms,
    ↪ delay_ms);

lv_style_set_transition(&style1, &trans1);
```

LV\_  
0,

## Color filter

TODO

## 主题 (Themes)

主题是样式的集合，如果启用主题，LVGL 会将其应用到每个部件（widgets），这将为 UI 提供默认的外观，然后可以通过添加更多样式进行修改。

每个显示器都可以有不同的主题。例如，您可以在 TFT 上使用彩色主题，在单色显示器上使用单色主题。

要使用主题，需要进行两个设置：

1. 初始化主题 Initialize a theme
2. 将主题应用到对应的显示器 Assign the initialized theme to a display.

主题初始化函数可以有不同的原型。此示例显示如何设置“默认”主题：



```

lv_theme_t * th = lv_theme_default_init(display, /*Use the DPI, size, etc from this_
↪display*/

/* 主次颜色 */
LV_COLOR_PALETTE_BLUE, LV_COLOR_PALETTE_CYAN,

↪ /*Primary and secondary palette*/

/* 明暗主题 */
false, /*Light or dark mode*/
/* 字体 */
&lv_font_montserrat_10,
&lv_font_montserrat_14,
&lv_font_montserrat_18); /*Small, normal,
↪large fonts*/

lv_disp_set_theme(display, th); /* 将主题应用到对应的显示器 Assign the theme to the_
↪display*/

```

LVGL 内置了主题并存放在 `lv_conf.h` 中，通过定义 `LV_USE_THEME_DEFAULT 1` 来使能默认主题。

### 扩展主题 (Extending themes)

内置主题可以扩展，如果用户创建了自己的主题，LVGL 将会把自定义主题作为默认主题的子主题，可以设置任意数量的包含主题，如 default theme -> custom theme -> dark theme.

使用函数 `lv_theme_set_parent(new_theme, base_theme)` 来扩展主题，其中 `base_theme` 是父主题，`new_theme` 是子主题。

示例如下

### 例子 (Examples)

#### API

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## 1.4.4 样式属性 (Style properties)

### 大小与位置 (Size and position)

Properties related to size, position, alignment and layout of the objects. 与对象的大小、位置、对齐方式和布局相关的属性。

## 宽度 (width)

可以使用像素、百分比和 `LV_SIZE_CONTENT` 来设置对象的宽度。百分比值与父对象的内容区域的宽度相关。

## 最小宽度 (min\_width)

设置最小宽度。可以使用像素值和百分比值。百分比值与父对象的内容区域的宽度相关。

## 最大宽度 (max\_width)

设置最大宽度。可以使用像素值和百分比值。百分比值与父对象的内容区域的宽度相关。

## 高度 (height)

可以使用像素、百分比和 `LV_SIZE_CONTENT` 来设置对象的高度。百分比值与父对象的内容区域的宽度相关。

## 最小高度 (min\_height)

设置最小高度。可以使用像素值和百分比值。百分比值与父对象的内容区域的宽度相关。

## 最大高度 (max\_height)

设置最大高度。可以使用像素值和百分比值。百分比值与父对象的内容区域的宽度相关。

## x

设置对象的 x 坐标，并需要考虑对齐 `align` 的方式. 可以使用像素值和百分比值。百分比值与父对象的内容区域的宽度相关。

## y

设置对象的 y 坐标，并需要考虑对齐 `align` 的方式. 可以使用像素值和百分比值。百分比值与父对象的内容区域的宽度相关。

## 对齐方式 (align)

设置对齐方式，该对齐方式告诉应从父节点的哪个位置来作为 X 和 Y 坐标的原点。共有以下几种选项: `LV_ALIGN_DEFAULT`, `LV_ALIGN_TOP_LEFT/MID/RIGHT`, `LV_ALIGN_BOTTOM_LEFT/MID/RIGHT`, `LV_ALIGN_LEFT/RIGHT_MID`, `LV_ALIGN_CENTER`。

`LV_ALIGN_DEFAULT` 代表 `LV_ALIGN_TOP_LEFT`，即以左上角作为原点，`LV_ALIGN_TOP_RIGHT` 则是右上角作为原点。

**变换宽度 (transform\_width)**

此值会使对象变得更宽。可以使用像素和百分比 (`lv_pct(x)`) 值。百分比值相对于对象的宽度而言。

**变换高度 (transform\_height)**

此值会使对象在两侧变得更高。可以使用像素和百分比 (`lv_pct(x)`) 值。百分比值相对于对象的高度而言。

**变换 x 坐标 (translate\_x)**

沿 x 方向移动对象，在完成布局、对齐和其他定位之后使用。可以使用像素和百分比 (使用 `lv_pct(x)`) 值。百分比值相对于对象的宽度而言。

**变换 y 坐标 (translate\_y)**

沿 y 方向移动对象，在完成布局、对齐和其他定位之后使用。可以使用像素和百分比 (使用 `lv_pct(x)`) 值。百分比值相对于对象的高度而言。

**缩放 (transform\_zoom)**

与图片缩放类似，与对象的属性进行相乘。256(或者 `LV_IMG_ZOOM_NONE`) 是普通尺寸，128 则缩放到原来的一半，512 则是放大一倍，以此类推。

**变换角度 (transform\_angle)**

与旋转图片类似类似地旋转对象，1 代表 0.1 度，则 45 度 = 450

**填充 (Padding)**

用于描述父级与子级之间以及子级之间间距的属性。非常类似于 HTML 中的填充属性。

**顶部填充 (pad\_top)**

在顶部设置填充。它使这个方向的内容区域变得更小。

**底部填充 (pad\_bottom)**

在底部设置填充。它使这个方向的内容区域变得更小。

**左部填充 (pad\_left)**

在左部设置填充。它使这个方向的内容区域变得更小。

**右部填充 (pad\_right)**

在右部设置填充。它使这个方向的内容区域变得更小。

**行填充 (pad\_row)**

设置行之间的填充。在布局时使用。

**列填充 (pad\_column)**

设置列之间的填充。在布局时使用。

**背景 (Background)**

用于描述对象的背景颜色和图像的属性。

**背景颜色 (bg\_color)**

Set the background color of the object.

**bg\_opa**

设置背景的不透明度。0, LV\_OPA\_0 或 LV\_OPA\_TRANSP 意味着完全透明, 255, LV\_OPA\_100 或 LV\_OPA\_COVER 意味着不透明, 其他值或 LV\_OPA\_10、LV\_OPA\_20 等表示半透明。

**渐变色 (bg\_grad\_color)**

设置背景的渐变色。只有在 grad\_dir 不是 LV\_GRAD\_DIR\_NONE 时生效。

**渐变方向 (bg\_grad\_dir)**

设置背景渐变的方向。有三个值 LV\_GRAD\_DIR\_NONE/HOR/VER。

**背景色起点 (bg\_main\_stop)**

设置背景色的起点。0 表示顶部/左侧，255 表示底部/右侧，128 表示中心，依此类推

**背景渐变色起点 (bg\_grad\_stop)**

设置背景渐变颜色的起点。0 表示顶部/左侧，255 表示底部/右侧，128 表示中心，依此类推

**背景图片 (bg\_img\_src)**

设置背景图像。可以是指向 lv\_img\_dsc\_t 的指针、文件路径或 LV\_SYMBOL\_...

**背景图片透明度 (bg\_img\_opa)**

设置背景图片的不透明度。0, LV\_OPA\_0 或 LV\_OPA\_TRANSP 意味着完全透明, 256, LV\_OPA\_100 或 LV\_OPA\_COVER 意味着不透明, 其他值或 LV\_OPA\_10、LV\_OPA\_20 等表示半透明。

**背景图片重新着色 (bg\_img\_recolor)**

设置一种颜色以混合到背景图像。

**背景图片重新着色强度 (bg\_img\_recolor\_opa)**

设置背景图像重新着色的强度。值 0、LV\_OPA\_0 或 LV\_OPA\_TRANSP 表示不混合，256、LV\_OPA\_100 或 LV\_OPA\_COVER 表示完全重新着色，其他值或 LV\_OPA\_10、LV\_OPA\_20 等按比例解释。

**背景图片平铺 (bg\_img\_tiled)**

如果启用，背景图像将被平铺。可能的值为 “true” 或 “false”。

**边框 (Border)**

描述边框的属性

**边框颜色 (border\_color)**

Set the color of the border

### 边框不透明度 (border\_opa)

设置边框的不透明度。0 , LV\_OPA\_0 或 LV\_OPA\_TRANSP 意味着完全透明, 256 , LV\_OPA\_100 或 LV\_OPA\_COVER 意味着不透明, 其他值或 LV\_OPA\_10、LV\_OPA\_20 等表示半透明。

### 边框宽度 (border\_width)

设置边框的宽度。只能使用像素值。

### 边界尺寸 (border\_side)

Set only which side(s) the border should be drawn. The possible values are LV\_BORDER\_SIDE\_NONE/TOP/BOTTOM/LEFT/RIGHT/INTERNAL. OR-ed values can be used as well, e.g. LV\_BORDER\_SIDE\_TOP | LV\_BORDER\_SIDE\_LEFT.

### border\_post

Sets whether the border should be drawn before or after the children are drawn. **true**: after children, **false**: before children

### 轮廓 (Outline)

描述轮廓的属性。它像边框，但绘制在矩形之外。

### 轮廓宽度 (outline\_width)

Set the width of the outline in pixels.

### outline\_color

Set the color of the outline.

### outline\_opa

Set the opacity of the outline. Value 0, LV\_OPA\_0 or LV\_OPA\_TRANSP means fully transparent, 256, LV\_OPA\_100 or LV\_OPA\_COVER means fully covering, other values or LV\_OPA\_10, LV\_OPA\_20, etc means semi transparency.

### **outline\_pad**

Set the padding of the outline, i.e. the gap between object and the outline.

### **Shadow**

Properties to describe the shadow drawn under the rectangles.

#### **shadow\_width**

Set the width of the shadow in pixels. The value should be  $\geq 0$ .

#### **shadow\_ofs\_x**

Set an offset on the shadow in pixels in X direction.

#### **shadow\_ofs\_y**

Set an offset on the shadow in pixels in Y direction.

#### **shadow\_spread**

Make the shadow calculation to use a larger or smaller rectangle as base. The value can be in pixel to make the area larger/smaller

#### **shadow\_color**

Set the color of the shadow

#### **shadow\_opa**

Set the opacity of the shadow. Value 0, LV\_OPA\_0 or LV\_OPA\_TRANSP means fully transparent, 256, LV\_OPA\_100 or LV\_OPA\_COVER means fully covering, other values or LV\_OPA\_10, LV\_OPA\_20, etc means semi transparency.

### **Image**

Properties to describe the images

## **img\_opa**

Set the opacity of an image. Value 0, LV\_OPA\_0 or LV\_OPA\_TRANSP means fully transparent, 256, LV\_OPA\_100 or LV\_OPA\_COVER means fully covering, other values or LV\_OPA\_10, LV\_OPA\_20, etc means semi transparency.

## **img\_recolor**

Set color to mix to the image.

## **img\_recolor\_opa**

Set the intensity of the color mixing. Value 0, LV\_OPA\_0 or LV\_OPA\_TRANSP means fully transparent, 256, LV\_OPA\_100 or LV\_OPA\_COVER means fully covering, other values or LV\_OPA\_10, LV\_OPA\_20, etc means semi transparency.

## **Line**

Properties to describe line-like objects

### **line\_width**

Set the width of the lines in pixel.

### **line\_dash\_width**

Set the width of dashes in pixel. Note that dash works only on horizontal and vertical lines

### **line\_dash\_gap**

Set the gap between dashes in pixel. Note that dash works only on horizontal and vertical lines

### **line\_rounded**

Make the end points of the lines rounded. **true**: rounded, **false**: perpendicular line ending

### **line\_color**

Set the color for the lines.



**line\_opa**

Set the opacity of the lines.

**Arc**

TODO

**arc\_width**

Set the width (thickness) of the arcs in pixel.

**arc\_rounded**

Make the end points of the arcs rounded. `true`: rounded, `false`: perpendicular line ending

**arc\_color**

Set the color of the arc.

**arc\_opa**

Set the opacity of the arcs.

**arc\_img\_src**

Set an image from which the arc will be masked out. It's useful to display complex effects on the arcs. Can be a pointer to `lv_img_dsc_t` or a path to a file

**Text**

Properties to describe the properties of text. All these properties are inherited.

**text\_color**

Sets the color of the text.

**text\_opa**

Set the opacity of the text. Value 0, LV\_OPA\_0 or LV\_OPA\_TRANSP means fully transparent, 256, LV\_OPA\_100 or LV\_OPA\_COVER means fully covering, other values or LV\_OPA\_10, LV\_OPA\_20, etc means semi transparency.

**text\_font**

Set the font of the text (a pointer lv\_font\_t \*).

**text\_letter\_space**

Set the letter space in pixels

**text\_line\_space**

Set the line space in pixels.

**text\_decor**

Set decoration for the text. The possible values are LV\_TEXT\_DECOR\_NONE/UNDERLINE/STRIKETHROUGH. OR-ed values can be used as well.

**text\_align**

Set how to align the lines of the text. Note that it doesn't align the object itself, only the lines inside the object. The possible values are LV\_TEXT\_ALIGN\_LEFT/CENTER/RIGHT/AUTO. LV\_TEXT\_ALIGN\_AUTO detect the text base direction and uses left or right alignment accordingly

**Miscellaneous**

Mixed proprites for various purposes.

**radius**

Set the radius on every corner. The value is interpreted in pixel ( $\geq 0$ ) or LV\_RADIUS\_CIRCLE for max. radius

**clip\_corner**

Enable to clip the overflowed content on the rounded corner. Can be true or false.

**opa**

Scale down all opacity values of the object by this factor. Value 0, LV\_OPA\_0 or LV\_OPA\_TRANSP means fully transparent, 256, LV\_OPA\_100 or LV\_OPA\_COVER means fully covering, other values or LV\_OPA\_10, LV\_OPA\_20, etc means semi transparency.

**color\_filter\_dsc**

Mix a color to all colors of the object.

**color\_filter\_opa**

The intensity of mixing of color filter.

**anim\_time**

The animation time in milliseconds. It's meaning is widget specific. E.g. blink time of the cursor on the text area or scroll time of a roller. See the widgets' documentation to learn more.

**anim\_speed**

The animation speed in pixel/sec. It's meaning is widget specific. E.g. scroll speed of label. See the widgets' documentation to learn more.

**transition**

An initialized lv\_style\_transition\_dsc\_t to describe a transition.

**blend\_mode**

Describes how to blend the colors to the background. The possible values are LV\_BLEND\_MODE\_NORMAL/ADDITIVE/SUBTRACTIVE/MULTIPLY

**layout**

Set the layout of the object. The children will be repositioned and resized according to the policies set for the layout. For the possible values see the documentation of the layouts.

## base\_dir

Set the base direction of the object. The possible values are LV\_BIDI\_DIR\_LTR/RTL/AUTO.

## 1.4.5 滚动 (Scroll)

### 概览 (Overview)

在 LVGL 中，滚动的工作非常直观：如果一个对象在其父内容区域（没有填充的大小）之外，则父对象变为可滚动并且会出现滚动条。

任何对象都可以滚动，包括 lv\_obj\_t, lv\_img, lv\_btn, lv\_meter, 等等。

对象可以一次水平或垂直滚动，但是不能对角。

### 滚动条 (Scrollbar)

#### 模式 (Mode)

滚动条根据配置的“模式”显示。存在以下“模式”：

- LV\_SCROLLBAR\_MODE\_OFF 不显示滚动条
- LV\_SCROLLBAR\_MODE\_ON 显示滚动条
- LV\_SCROLLBAR\_MODE\_ACTIVE 在滚动时显示滚动条
- LV\_SCROLLBAR\_MODE\_AUTO 当内容足够大可以滚动时显示滚动条

可以通过函数 lv\_obj\_set\_scrollbar\_mode(obj, LV\_SCROLLBAR\_MODE\_...) 在对象上设置滚动条模式。

#### 样式 (Styling)

滚动条有自己的专用的部件块 LV\_PART\_SCROLLBAR。例如，滚动条可以像这样变成红色：

```
static lv_style_t style_red;
lv_style_init(&style_red);
lv_style_set_bg_color(&style_red, lv_color_red());

...

lv_obj_add_style(obj, &style_red, LV_PART_SCROLLBAR);
```

对象在滚动时进入 LV\_STATE\_SCROLLED 状态。这允许在滚动时向滚动条或对象本身添加不同的样式。

例如，当对象滚动时，此代码使滚动条变为蓝色：

```
static lv_style_t style_blue;
lv_style_init(&style_blue);
lv_style_set_bg_color(&style_blue, lv_color_blue());

...

lv_obj_add_style(obj, &style_blue, LV_STATE_SCROLLED | LV_PART_SCROLLBAR);
```

如果 `LV_PART_SCROLLBAR` 的基本方向是 `RTL` (`LV_BASE_DIR_RTL`) (向右滚动), 则垂直滚动条将放置在左侧。请注意, `base_dir` 样式属性是继承的。因此, 它可以直接设置在对象的 `LV_PART_SCROLLBAR` 部分, 或在对象或任何父级的主要部分上使滚动条继承基本方向。

## 时间 (Events)

以下事件与滚动相关:

- `LV_EVENT_SCROLL_BEGIN` 开始滚动
- `LV_EVENT_SCROLL_END` 结束滚动
- `LV_EVENT_SCROLL` 正在滚动, 每次位置变化时触发。

## Basic example

TODO

## 滚动的特点 (Features of scrolling)

滚动有许多有用的附加功能。

## 可滚动性 (Scrollable)

可以通过函数 `lv_obj_clear_flag(obj, LV_OBJ_FLAG_SCROLLABLE)` 使对象不可滚动。

不可滚动的对象仍然可以将滚动 (链) 传播到它们的父对象。

滚动发生的方向可以通过函数 `lv_obj_set_scroll_dir(obj, LV_DIR_...)` 控制。

方向可能有以下值:

- `LV_DIR_TOP` 只向上
- `LV_DIR_LEFT` 只向左
- `LV_DIR_BOTTOM` 只向下
- `LV_DIR_RIGHT` 只向右
- `LV_DIR_HOR` 水平滚动
- `LV_DIR_TOP` 垂直滚动
- `LV_DIR_ALL` 随意滚动

同样可以使用或表达式: `LV_DIR_TOP | LV_DIR_LEFT`。

## 滚动链 (Scroll chain)

如果对象无法进一步滚动（例如其内容已到达最底部的位置），则额外的滚动会传播到其父对象。如果父级可以在那个方向滚动，那么它将被滚动，同样的，它将继续传播给祖父母级和祖祖父母级。

滚动传播称为“滚动链接”，可以使用 `LV_OBJ_FLAG_SCROLL_CHAIN` 标志来启用/禁用。

如果禁用滚动链接，则传播将停止在对象上，并且不会传递到父对象上。

## 滚动动量 (Scroll momentum)

当用户滚动对象并释放它时，LVGL 可以模拟滚动的惯性动量，类似于对象被抛出并且滚动缓慢减慢的效果。

可以使用 `LV_OBJ_FLAG_SCROLL_MOMENTUM` 标志启用/禁用滚动动量。

## 弹性滚动 (Elastic scroll)

通常，对象不能滚动超过其内容的末端。即内容的顶部不能低于对象的顶部。

但是，使用 `LV_OBJ_FLAG_SCROLL_ELASTIC` 标志会在用户“滚动”内容时添加一种奇特的效果。滚动变慢，内容可以在对象内部滚动，当对象被释放时，滚动到其中的内容将通过动画的形式回到有效位置。

## 捕捉 (Snapping)

滚动结束时，可以根据特定规则捕捉对象的子项。可以使用 `LV_OBJ_FLAG_SNAPPABLE` 标志将子对象单独设置为可捕捉。

对象可以通过四种方式捕捉子对象：

- `LV_SCROLL_SNAP_NONE` 禁用捕捉（默认）
- `LV_SCROLL_SNAP_START` 将子对象与滚动对象的左侧/顶部对齐
- `LV_SCROLL_SNAP_END` 将子对象与滚动对象的右侧/底部对齐
- `LV_SCROLL_SNAP_CENTER` 将子对象与滚动对象的中心对齐

捕捉对齐方式通过函数 `lv_obj_set_scroll_snap_x/y(obj, LV_SCROLL_SNAP_...)` 来设置：

滚动在底层中具体的实现过程如下：

1. 用户滚动对象并释放屏幕
2. LVGL 考虑滚动动量计算滚动结束的位置
3. LVGL 寻找最近的滚动点
4. LVGL 滚动到带动画的捕捉点

## 单次滚动 (Scroll one)

单次滚动即 LVGL 只允许在滚动中捕捉一次子对象，这需要使子对象可捕捉并设置与 LV\_SCROLL\_SNAP\_NONE 不同的滚动对齐方式。

此功能可以通过 LV\_OBJ\_FLAG\_SCROLL\_ONE 标志启用。

## Scroll on focus

Imagine that there a lot of objects in a group that are on a scrollable object. Pressing the “Tab” button focuses the next object but it might be outside the visible area of the scrollable object. If the “scroll on focus” feature is enabled LVGL will automatically scroll objects to bring their children into view. The scrolling happens recursively therefore even nested scrollable objects are handled properly. The object will be scrolled into view even if it's on a different page of a tabview.

## Scroll manually

The following API functions allow manual scrolling of objects:

- `lv_obj_scroll_by(obj, x, y, LV_ANIM_ON/OFF)` scroll by x and y values
- `lv_obj_scroll_to(obj, x, y, LV_ANIM_ON/OFF)` scroll to bring the given coordinate to the top left corner
- `lv_obj_scroll_to_x(obj, x, LV_ANIM_ON/OFF)` scroll to bring the given coordinate to the left side
- `lv_obj_scroll_to_y(obj, y, LV_ANIM_ON/OFF)` scroll to bring the given coordinate to the top side

## Self size

Self size is a property of an object. Normally, the user shouldn't use this parameter but if a custom widget is created it might be useful.

In short, self size establishes the size of an object's content. To understand it better take the example of a table. Let's say it has 10 rows each with 50 px height. So the total height of the content is 500 px. In other words the “self height” is 500 px. If the user sets only 200 px height for the table LVGL will see that the self size is larger and make the table scrollable.

This means not only the children can make an object scrollable but a larger self size will too.

LVGL uses the LV\_EVENT\_GET\_SELF\_SIZE event to get the self size of an object. Here is an example to see how to handle the event:

```
if(event_code == LV_EVENT_GET_SELF_SIZE) {
    lv_point_t * p = lv_event_get_param(e);

    //If x or y < 0 then it doesn't need to be calculated now
    if(p->x >= 0) {
        p->x = 200;           //Set or calculate the self width
    }

    if(p->y >= 0) {
        p->y = 50;           //Set or calculate the self height
    }
}
```

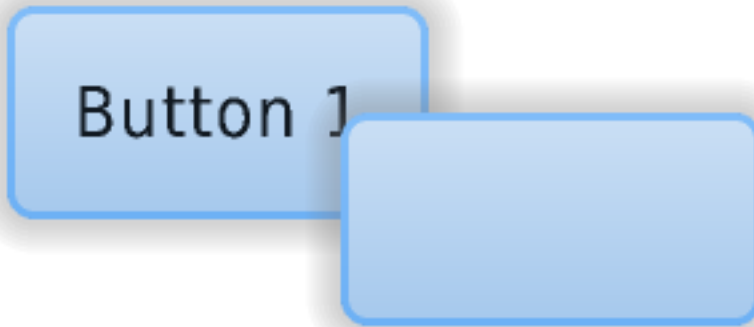
## Examples

### 1.4.6 层 (Layers)

#### 创建顺序 (Order of creation)

默认情况下，LVGL 在旧对象之上绘制新对象。

例如，假设我们将一个按钮添加到名为 `button1` 的父对象，然后再添加另一个名为 `button2` 的按钮。然后 `button1`（及其子对象）将位于背景中，并且可以被 `button2` 及其子对象覆盖。



```

/*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 automatically setting
↳the size according to 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*/

```

(下页继续)



(续上页)

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

### 将对象置于上层 (Bring to the foreground)

有四种显式方法可以改变对象所在的层：

- 函数 `lv_obj_move_foreground(obj)` 将对象置于最上层，同样的，使用函数 `lv_obj_move_background(obj)` 将对象置于最下层。
- 函数 `lv_obj_move_up(obj)` 将对象向上移动一层，同样的，使用函数 `lv_obj_move_down(obj)` 将对象向下移动一层。
- 函数 `lv_obj_swap(obj1, obj2)` 将两个对象的层交换。
- 函数 `lv_obj_set_parent(obj, new_parent)` 将对象 `obj` 置于 `new_parent` 之上。

### 顶层与系统层 (Top and sys layers)

LVGL 使用名为 “layer\_top” 和 “layer\_sys” 的两个特殊层。两者在显示器的所有屏幕上都是可见的和通用的。但是，它们不会在多个物理显示器之间共享。

`layer_top` 始终位于默认屏幕 (`lv_scr_act()`) 的上一层，而 `layer_sys` 位于 `layer_top` 的上一层。

用户可以使用 `layer_top` 来创建一些可见的内容。例如，一个菜单栏、一个弹出窗口等。如果启用了 `click` 属性，那么 `layer_top` 将吸收所有用户点击并充当模态。

```
lv_obj_add_flag(lv_layer_top(), LV_OBJ_FLAG_CLICKABLE);
```

`layer_sys` 在 LVGL 中也用于类似的目的。例如，它将鼠标光标放在所有图层上方以确保它始终可见。

## 1.4.7 事件 (Events)

当对象被操作时，LVGL 中会触发事件，例如当一个对象

- 被点击
- 滚动
- 改变值
- 重绘等

### 给对象添加事件 (Add events to the object)

用户可以为对象分配回调函数以查看其事件，比如：

```
lv_obj_t * btn = lv_btn_create(lv_scr_act());
lv_obj_add_event_cb(btn, my_event_cb, LV_EVENT_CLICKED, NULL); /* 注册回调函数 Assign
↪ an event callback*/
...

```

(下页继续)

(续上页)

```
static void my_event_cb(lv_event_t * event)
{
    printf("Clicked\n");
}
```

在这个例子中，LV\_EVENT\_CLICKED 表示只有在对象被点击时才会触发回调函数 my\_event\_cb。具体请看 [list of event codes](#) LV\_EVENT\_ALL 则表示接受所有事件。

lv\_obj\_add\_event\_cb 的最后一个参数是指向事件中可用的回调函数的指针，稍后将更详细地描述。

可以向一个对象添加多个事件，如下所示：

```
lv_obj_add_event_cb(obj, my_event_cb_1, LV_EVENT_CLICKED, NULL);
lv_obj_add_event_cb(obj, my_event_cb_2, LV_EVENT_PRESSED, NULL);
lv_obj_add_event_cb(obj, my_event_cb_3, LV_EVENT_ALL, NULL);
```

/\*No filtering, receive all events\*/

即使是相同事件的处理也可以用于具有不同 user\_data 的回调函数。例如：

```
lv_obj_add_event_cb(obj, increment_on_click, LV_EVENT_CLICKED, &num1);
lv_obj_add_event_cb(obj, increment_on_click, LV_EVENT_CLICKED, &num2);
```

这些事件将按照添加的顺序被调用，其他对象可以使用相同的回调函数。

### 从对象中删除事件 (Remove event(s) from an object)

使用函数 lv\_obj\_remove\_event\_cb(obj, event\_cb) 或者 lv\_obj\_remove\_event\_dsc(obj, event\_dsc) 来删除对象中的事件。event\_dsc 是在执行函数 lv\_obj\_add\_event\_cb 时所用的参数。

### 事件代码 (Event codes)

事件代码可以分为以下几类：

- 输入设备事件 Input device events
- 绘图事件 Drawing events
- 其他事件 Other events
- 特殊事件 Special events
- 自定义事件 Custom events

所有对象（例如按钮/标签/滑块等），无论其类型如何，都会接收 *Input device*、*Drawing* 和 *Other* 事件。

特殊事件只存在与某些部件类型，具体请看 [widgets' documentation](#)

自定义事件由用户添加，LVGL 不会产生此事件。

LVGL 定义了如下事件代码：

## Input device events

- **LV\_EVENT\_PRESSED** 对象被按下
- **LV\_EVENT\_PRESSING** 正在按下一个对象（按下时连续调用）
- **LV\_EVENT\_PRESS\_LOST** 一个对象仍在被按下，但光标/手指已离开该对象（松手）
- **LV\_EVENT\_SHORT\_CLICKED** 一个对象被压了一小段时间，然后被释放。如果是在滚动则不会调用。
- **LV\_EVENT\_LONG\_PRESSED** 至少在输入设备驱动程序中指定的 `long_press_time` 时间内按下了一个对象。如果是在滚动则不会调用。
- **LV\_EVENT\_LONG\_PRESSED\_REPEAT** 在 每个 `long_press_repeat_time` 毫秒的 `long_press_time` 之后调用。如果是在滚动则不会调用。
- **LV\_EVENT\_CLICKED** 如果对象没有滚动，则在释放时调用（无论是否长按）
- **LV\_EVENT\_RELEASED** 当一个对象被释放时，在任何情况下都调用
- **LV\_EVENT\_SCROLL\_BEGIN** 滚动开始。事件参数是 `NULL` 或带有滚动动画描述符的 `lv_anim_t *`，如果需要可以修改。
- **LV\_EVENT\_SCROLL\_END** 滚动结束。
- **LV\_EVENT\_SCROLL** 对象滚动过了
- **LV\_EVENT\_GESTURE** 检测到手势。通过函数获取手势 `lv_indev_get_gesture_dir(lv_indev_get_act());`
- **LV\_EVENT\_KEY** A key is sent to an object. Get the key with `lv_indev_get_key(lv_indev_get_act());`
- **LV\_EVENT\_FOCUSED** 一个对象被聚焦
- **LV\_EVENT\_DEFOCUSED** 对象未聚焦
- **LV\_EVENT\_LEAVE** 对象未聚焦但仍被选中
- **LV\_EVENT\_HIT\_TEST** 执行高级命中测试。使用 `lv_hit_test_info_t * a = lv_event_get_hit_test_info(e)` 并且检查 `a->point` 是否可以点击对象，如果没有设置则 `a->res = false`

## 绘图事件 (Drawing events)

- **LV\_EVENT\_COVER\_CHECK** 检查物体是否完全覆盖了一个区域。事件参数是 `lv_cover_check_info_t *`。
- **LV\_EVENT\_REFR\_EXT\_DRAW\_SIZE** Get the required extra draw area around an object (e.g. for a shadow). The event parameter is `lv_coord_t *` to store the size. Only overwrite it with a larger value.
- **LV\_EVENT\_DRAW\_MAIN\_BEGIN** Starting the main drawing phase.
- **LV\_EVENT\_DRAW\_MAIN** Perform the main drawing
- **LV\_EVENT\_DRAW\_MAIN\_END** Finishing the main drawing phase
- **LV\_EVENT\_DRAW\_POST\_BEGIN** Starting the post draw phase (when all children are drawn)
- **LV\_EVENT\_DRAW\_POST** Perform the post draw phase (when all children are drawn)
- **LV\_EVENT\_DRAW\_POST\_END** Finishing the post draw phase (when all children are drawn)
- **LV\_EVENT\_DRAW\_PART\_BEGIN** Starting to draw a part. The event parameter is `lv_obj_draw_dsc_t *`. Learn more [here](#).

- LV\_EVENT\_DRAW\_PART\_END Finishing to draw a part. The event parameter is `lv_obj_draw_dsc_t *`. Learn more [here](#).

### 其他事件 (Other events)

- LV\_EVENT\_DELETE 删除对象
- LV\_EVENT\_CHILD\_CHANGED 添加/删除子对象
- LV\_EVENT\_CHILD\_CREATED 创建了新子对象，并通知所有父对象
- LV\_EVENT\_CHILD\_DELETED 删除了新子对象，并通知所有父对象
- LV\_EVENT\_SIZE\_CHANGED 对象坐标/大小已更改
- LV\_EVENT\_STYLE\_CHANGED 对象的样式已更改
- LV\_EVENT\_BASE\_DIR\_CHANGED 基本目录已更改
- LV\_EVENT\_GET\_SELF\_SIZE 获取部件块的内部大小
- LV\_EVENT\_SCREEN\_UNLOAD\_START 屏幕开始删除，当 `lv_scr_load/lv_scr_load_anim` 被调用时立即触发
- LV\_EVENT\_SCREEN\_LOAD\_START 屏幕加载开始，当屏幕更改延迟到期时触发
- LV\_EVENT\_SCREEN\_LOADED 屏幕已加载，在所有动画完成时调用
- LV\_EVENT\_SCREEN\_UNLOADED 屏幕已删除，在所有动画完成时调用

### 特殊事件 (Special events)

- LV\_EVENT\_VALUE\_CHANGED 对象的值已更改（如滑块移动）
- LV\_EVENT\_INSERT 文本被插入到对象中。事件数据是插入的 `char *`。
- LV\_EVENT\_REFRESH 通知对象刷新其上的某些内容（对于用户）
- LV\_EVENT\_READY 一个进程已经完成
- LV\_EVENT\_CANCEL 一个进程被取消

### 自定义事件 (Custom events)

Any custom event codes can be registered by `uint32_t MY_EVENT_1 = lv_event_register_id();`

They can be sent to any object with `lv_event_send(obj, MY_EVENT_1, &some_data)`

### 手动发送事件 (Sending events)

要手动向对象发送事件，请使用 `lv_event_send(obj, <EVENT_CODE> &some_data)`。

例如，这可用于通过模拟按钮按下来手动关闭消息框（尽管有更简单的方法可以做到这一点）：

```
/*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);
```

## 刷新事件 (Refresh event)

LV\_EVENT\_REFRESH 是一个特殊事件，因为它旨在让用户通知对象刷新自身。如：

- 通知标签根据一个或多个变量（例如当前时间）刷新其文本
- 当语言改变时刷新标签
- 如果满足某些条件（例如输入正确的 PIN），则启用按钮
- 如果超出限制，则向/从对象添加/删除样式等

## Fields of lv\_event\_t

lv\_event\_t is the only parameter passed to the event callback and it contains all data about the event. The following values can be gotten from it:

- lv\_event\_get\_code(e) get the event code
- lv\_event\_get\_current\_target(e) get the object to which an event was sent. I.e. the object whose event handler is being called.
- lv\_event\_get\_target(e) get the object that originally triggered the event (different from lv\_event\_get\_target if *event bubbling* is enabled)
- lv\_event\_get\_user\_data(e) get the pointer passed as the last parameter of lv\_obj\_add\_event\_cb.
- lv\_event\_get\_param(e) get the parameter passed as the last parameter of lv\_event\_send

## Event bubbling

If lv\_obj\_add\_flag(obj, LV\_OBJ\_FLAG\_EVENT\_BUBBLE) is enabled all events will be sent to an object's parent too. If the parent also has LV\_OBJ\_FLAG\_EVENT\_BUBBLE enabled the event will be sent to its parent and so on.

The *target* parameter of the event is always the current target object, not the original object. To get the original target call lv\_event\_get\_original\_target(e) in the event handler.

## Examples

### 1.4.8 输入设备 (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

---

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

---

## Pointers

### Cursor

Pointer input devices (like a mouse) can have a cursor.

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

LV_IMG_DECLARE(mouse_cursor_icon);           /*Declare the image_
↪source.*/
lv_obj_t * cursor_obj = lv_img_create(lv_scr_act()); /*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_clear_flag(cursor_obj, LV_OBJ_FLAG_CLICKABLE)`. For images, *clicking* is disabled by default.

### Gestures

Pointer input devices can detect basic gestures. By default, most of the widgets send the gestures to its parent, so finally the gestures can be detected on the screen object in a form of an `LV_EVENT_GESTURE` event. For example:

```
void my_event(lv_event_t * e)
{
    lv_obj_t * screen = lv_event_get_current_target(e);
    lv_dir_t dir = lv_indev_get_gesture_dir(lv_indev_act());
    switch(dir) {
        case LV_DIR_LEFT:
            ...
            break;
        case LV_DIR_RIGHT:
            ...
            break;
        case LV_DIR_TOP:
            ...
            break;
        case LV_DIR_BOTTOM:
            ...
            break;
    }
}

...

lv_obj_add_event_cb(screen1, my_event, LV_EVENT_GESTURE, NULL);
```

To prevent passing the gesture event to the parent from an object use `lv_obj_clear_flag(obj, LV_OBJ_FLAG_GESTURE_BUBBLE)`.

## Keypad and encoder

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

## Groups

Objects you want to control with a keypad or encoder need 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 key events to only one group but a group can receive data from more than one input device.

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 to the right
- **LV\_KEY\_LEFT** Decrease value or move to 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 support navigation in a group and interact with selected objects.

Usually, it's enough to use only `LV_KEY_LEFT/RIGHT` because most 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 objects and edit them using the keypad. But encoders have a limited number of “keys” and hence it is difficult to navigate using the default options. *Navigate* and *Edit* modes are used to avoid this problem with encoders.

In *Navigate* mode, an encoder's 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 modify an 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 cannot be pressed (like a *Slider*) leaves *Edit* mode upon a short click. But with objects where a short click has meaning (e.g. *Button*), a long press is required.

## Default group

Interactive widgets - such as buttons, checkboxes, sliders, etc. - can be automatically added to a default group. Just create a group with `lv_group_t * g = lv_group_create();` and set the default group with `lv_group_set_default(g);`

Don't forget to assign one or more input devices to the default group with `lv_indev_set_group(my_indev, g);`.

## Styling

If an object is focused either by clicking it via touchpad or focused via an encoder or keypad it goes to the LV\_STATE\_FOCUSED state. Hence, focused styles will be applied to it.

If an object switches to edit mode it enters the LV\_STATE\_FOCUSED | LV\_STATE\_EDITED states so these style properties will be shown.

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

## API

### Input device

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Groups

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

### 1.4.9 显示器 (Displays)

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



## 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 the 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(display)` 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 handling is completely hidden if you register only one display. By default, the last created (and only) display is used.

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

## Mirror display

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

## Split image

You can create a larger virtual display from an array of 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 real display with the truncated area.

## Screens

Every display has its own set of [screens](#) and the objects on each screen.

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. A screen's size is always equal to its display and their origin is (0;0). Therefore, a screen's 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 *Base object* and *Image* (to create a wallpaper).

To create a screen, use `lv_obj_t * scr = lv_<type>_create(NULL, copy)`. `copy` can be an existing screen copied into the new screen.

To load a screen, use `lv_scr_load(scr)`. To get the active screen, use `lv_scr_act()`. These functions work on the default display. If you want to specify which display to work on, use `lv_disp_get_scr_act(display)` and `lv_disp_load_scr(display, scr)`. A 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 this is 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 on a lower layer, and a menu is overlaid on an upper layer.

To handle transparent displays, special (slower) color mixing algorithms need 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 255 where there are solid objects.

In summary, to enable transparent screens and displays for 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_OBJMASK_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);`

## Features of displays

### Inactivity

A user's inactivity time 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(display)`. If `NULL` is passed, the lowest inactivity time among all displays will be returned (**NULL isn't just the default display**).

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

## Background

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

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

The display background image is a path to a file or a pointer to an `lv_img_dsc_t` variable (converted image data) to be used as wallpaper. It can be set with `lv_disp_set_bg_image(disp, &my_img);` If a background image is configured 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(disp, opa)`.

The `disp` parameter of these functions can be `NULL` to select the default display.

## API

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

### 1.4.10 色彩 (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 type `lv_color_t` is used to store a color. Its fields are set according to `LV_COLOR_DEPTH` in `lv_conf.h`. (See below)

You may set `LV_COLOR_16_SWAP` in `lv_conf.h` to swap bytes of `RGB565` colors. You may need this when sending 16-bit colors via a byte-oriented interface like SPI. As 16-bit numbers are stored in little-endian format (lower byte at 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.

## Creating colors

### RGB

Create colors from Red, Green and Blue channel values:

```
//All channels are 0-255
lv_color_t c = lv_color_make(red, green, blue);

//From hex code 0x000000..0xFFFFFF interpreted as RED + GREEN + BLUE
lv_color_t c = lv_color_hex(0x123456);

//From 3 digits. Same as lv_color_hex(0x112233)
lv_color_t c = lv_color_hex3(0x123);
```

## HSV

Create colors from Hue, Saturation and Value values:

```
//h = 0..359, s = 0..100, v = 0..100
lv_color_t c = lv_color_hsv_to_rgb(h, s, v);

//All channels are 0-255
lv_color_hsv_t c_hsv = lv_color_rgb_to_hsv(r, g, b);

//From lv_color_t variable
lv_color_hsv_t c_hsv = lv_color_to_hsv(color);
```

## Palette

LVGL includes [Material Design's palette](#) of colors. In this system all named colors have a nominal main color as well as four darker and five lighter variants.

The names of the colors are as follows:

- LV\_PALETTE\_RED
- LV\_PALETTE\_PINK
- LV\_PALETTE\_PURPLE
- LV\_PALETTE\_DEEP\_PURPLE
- LV\_PALETTE\_INDIGO
- LV\_PALETTE\_BLUE
- LV\_PALETTE\_LIGHT\_BLUE
- LV\_PALETTE\_CYAN
- LV\_PALETTE\_TEAL
- LV\_PALETTE\_GREEN
- LV\_PALETTE\_LIGHT\_GREEN
- LV\_PALETTE\_LIME
- LV\_PALETTE\_YELLOW
- LV\_PALETTE\_AMBER
- LV\_PALETTE\_ORANGE
- LV\_PALETTE\_DEEP\_ORANGE
- LV\_PALETTE\_BROWN
- LV\_PALETTE\_BLUE\_GREY
- LV\_PALETTE\_GREY

To get the main color use `lv_color_t c = lv_palette_main(LV_PALETTE_...)`.

For the lighter variants of a palette color use `lv_color_t c = lv_palette_lighten(LV_PALETTE_..., v)`. `v` can be 1..5. For the darker variants of a palette color use `lv_color_t c = lv_palette_darken(LV_PALETTE_..., v)`. `v` can be 1..4.

## Modify and mix colors

The following functions can modify a color:

```
// Lighten a color. 0: no change, 255: white
lv_color_t c = lv_color_lighten(c, lvl);

// Darken a color. 0: no change, 255: black
lv_color_t c = lv_color_darken(lv_color_t c, lv_opa_t lvl);

// Lighten or darken a color. 0: black, 128: no change 255: white
lv_color_t c = lv_color_change_lightness(lv_color_t c, lv_opa_t lvl);

// Mix two colors with a given ratio 0: full c2, 255: full c1, 128: half c1 and half c2
lv_color_t c = lv_color_mix(c1, c2, ratio);
```

## Built-in colors

`lv_color_white()` and `lv_color_black()` return `0xFFFFFFFF` and `0x000000` respectively.

## Opacity

To describe opacity the `lv_opa_t` type is created from `uint8_t`. Some special purpose defines are also introduced:

- `LV_OPA_TRANSP` Value: 0, means no opacity making the color completely transparent
- `LV_OPA_10` Value: 25, means the color covers only a little
- `LV_OPA_20` ... `OPA_80` follow logically
- `LV_OPA_90` Value: 229, means the color near completely covers
- `LV_OPA_COVER` Value: 255, means the color completely covers (full opacity)

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

## Color types

The following variable types are defined by the color module:

- `lv_color1_t` Monochrome color. Also has R, G, B fields for compatibility 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` depending on the configured color depth setting
- `lv_color_int_t` `uint8_t`, `uint16_t` or `uint32_t` depending on the 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 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 to map a converted color back into a structure:

```
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 c24;
c24.full = lv_color_to32(c);        /*Give a 32 bit number with the converted color*/
```

## API

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

### 1.4.11 字体 (Fonts)

在 LVGL 中, 字体是渲染单个字母 (字形) 图像所需的位图和其他信息的集合, 字体存储在 `lv_font_t` 变量中, 可以在样式的 `text_font` 字段中设置。例如:

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

字体具有 **bpp (像素位数)** 属性, 它显示了使用多少位来描述字体中的像素, 像素值决定像素的不透明度。这样, 使用更高的 *bpp*, 字母的边缘可以更平滑。可能的 *bpp* 值为 1、2、4 和 8 (值越高表示质量越好)。*bpp* 属性还会影响存储字体所需的内存量。例如, *bpp* = 4 使字体比 *bpp* = 1 大近四倍。

## Unicode 编码支持 (Unicode support)

LVGL 支持 **UTF-8** 编码的 Unicode 字符。编辑器通常需要配置为将代码/文本保存为 UTF-8 (通常是默认值) 格式, `LV_TXT_ENC` 在 `lv_conf.h` 中设置为 `LV_TXT_ENC_UTF8`。(默认值)

测试:

```
lv_obj_t * label1 = lv_label_create(lv_scr_act(), NULL);  
lv_label_set_text(label1, LV_SYMBOL_OK);
```

如果一切正常, 应显示 ✓ 字符。

## 内嵌字体 (Built-in fonts)

有几种不同大小的内置字体, 可以在 `lv_conf.h` 中使用 `LV_FONT_...` 定义启用。

## 普通字体 (Normal fonts)

包含所有 ASCII 字符、度数符号 (U+00B0)、子弹符号 (U+2022) 和内置符号 (见下文)。

- `LV_FONT_MONTERRAT_12` 12 px font
- `LV_FONT_MONTERRAT_14` 14 px font
- `LV_FONT_MONTERRAT_16` 16 px font
- `LV_FONT_MONTERRAT_18` 18 px font
- `LV_FONT_MONTERRAT_20` 20 px font
- `LV_FONT_MONTERRAT_22` 22 px font
- `LV_FONT_MONTERRAT_24` 24 px font
- `LV_FONT_MONTERRAT_26` 26 px font
- `LV_FONT_MONTERRAT_28` 28 px font
- `LV_FONT_MONTERRAT_30` 30 px font
- `LV_FONT_MONTERRAT_32` 32 px font
- `LV_FONT_MONTERRAT_34` 34 px font
- `LV_FONT_MONTERRAT_36` 36 px font
- `LV_FONT_MONTERRAT_38` 38 px font
- `LV_FONT_MONTERRAT_40` 40 px font
- `LV_FONT_MONTERRAT_42` 42 px font
- `LV_FONT_MONTERRAT_44` 44 px font
- `LV_FONT_MONTERRAT_46` 46 px font
- `LV_FONT_MONTERRAT_48` 48 px font

## 特殊字体 (Special fonts)


























































- `LV_FONT_MONTERRAT_12_SUBPX` 与普通 12 px 字体相同, 但具有 [子像素渲染] *subpixel rendering*
- `LV_FONT_MONTERRAT_28_COMPRESSED` 与普通 28 px 字体相同, 但存储为 3 bpp 字体 *compressed font*
- `LV_FONT_DEJAVU_16_PERSIAN_HEBREW` 16 px 字体, 正常范围 + 希伯来语、阿拉伯语、波斯语字母及其所有形式
- `LV_FONT_SIMSUN_16_CJK` 具有正常范围的 16 px 字体加上 1000 个最常见的 CJK 部首
- `LV_FONT_UNSCII_8` 8 px 像素完美字体, 只有 ASCII 字符
- `LV_FONT_UNSCII_16` 16 px 像素完美字体, 只有 ASCII 字符

内置字体是全局变量, 名称如下 `lv_font_montserrat_16` 对于 16 px 高度的字体。要在样式中使用它们, 只需添加一个指向字体变量的指针, 即字体名称。

*bpp* = 4 的内置字体包含 ASCII 字符并使用 *Montserrat* 字体。

除了 ASCII 范围外, 以下符号还添加到 *FontAwesome* 字体的内置字体中。



	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		

这些符号可以单独使用：

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

或与字符串一起使用（编译时字符串连接）：

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

或多个符号组合在一起：

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

## 特性 (Special features)

### 双向书写支持 (Bidirectional support)

大多数语言使用从左到右 (简称 LTR) 书写方向, 但是某些语言 (例如希伯来语、波斯语或阿拉伯语) 使用从右到左 (简称 RTL) 方向。

LVGL 不仅支持 RTL 文本, 还支持混合 (又名双向, BiDi) 文本渲染。一些例子:

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

The title is مفتاح معايير الويب!

BiDi 支持由 *lv\_conf.h* 中的 `LV_USE_BIDI` 启用

所有文本都有一个基本方向 (LTR 或 RTL), 它决定了一些渲染规则和文本的默认对齐方式 (左或右)。然而, 在 LVGL 中, 基本方向不仅适用于标签, 同样可以为每个对象设置的通用属性。如果未设置, 则它将从父级继承, 这意味着设置屏幕的基本方向就足够了, 每个对象都会继承它。

屏幕的默认基本方向可以通过 *lv\_conf.h* 中的 `LV_BIDI_BASE_DIR_DEF` 设置, 其他对象从其父对象继承方向。

要设置对象的基本方向, 请使用 `lv_obj_set_base_dir(obj, base_dir)`。可能的基本方向是:

- `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 base direction from the parent (or a default value for non-screen objects)

此列表总结了 RTL 基本方向对对象的影响:

- Create objects by default on the right
- `lv_tabview`: Displays tabs from right to left
- `lv_checkbox`: Shows the box on the right
- `lv_btnmatrix`: Shows buttons from right to left
- `lv_list`: Shows icons on the right
- `lv_dropdown`: Aligns 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

## 阿拉伯语和波斯语支持 (Arabic and Persian support)

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

LVGL supports these rules if `LV_USE_ARABIC_PERSIAN_CHARS` is enabled.

However, there some limitations:

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

## 亚像素渲染 (Subpixel rendering)

抗锯齿相关，若有需要请自行翻译

Subpixel rendering allows for tripling the horizontal resolution by rendering anti-aliased edges on Red, Green and Blue channels instead of at pixel level granularity. This takes advantage of the position of physical color channels of each pixel, resulting in higher quality letter anti-aliasing. Learn more [here](#).

For subpixel rendering, the fonts need to be generated 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 three 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, LVGL assumes RGB order, however this can be swapped by setting `LV_SUBPX_BGR 1` in `lv_conf.h`.

## 压缩字体 (Compressed fonts)

The bitmaps of fonts can be compressed by

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

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

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

## 增加新字体 (Add a new font)

有几种方法可以将新字体添加到项目中：

1. 最简单的方法是使用 [Online font converter](#). 只需设置参数，单击 *Convert* 按钮，将字体复制到项目中即可。请务必仔细阅读该站点上提供的步骤，否则在转换时会出错。
2. 使用 [Offline font converter](#). (需要安装 Node.js)
3. 如果您想创建类似与内置字体 (Montserrat 字体和符号)，但大小和/或范围不同的内容，您可以使用 `lvgl/scripts/built_in_font` 文件夹中的 `built_in_font_gen.py` 脚本。(需要安装 Python 和 `lv_font_conv`)

要在文件中声明字体，请使用 `LV_FONT_DECLARE(my_font_name)`。

要使字体全局可用（如内置字体），请将它们添加到 `lv_conf.h` 中的 `LV_FONT_CUSTOM_DECLARE`。

## 添加新符号 (Add new symbols)

内置符号是从 [FontAwesome](#) 字体创建的。

1. 在 <https://fontawesome.com> 上搜索符号。例如 USB 符号。复制其 Unicode ID，在本例中为 `0xf287`。
2. 打开 [Online font converter](#). 添加 `FontAwesome.woff`。
3. 设置名称、大小、BPP 等参数。将使用此名称在项目中声明和使用字体
4. 将符号的 Unicode ID 添加到范围字段。例如，USB 符号的 `0xf287`。更多的符号可以用，枚举。
5. 转换字体并将生成的源代码复制到您的项目中，需要确保编译了字体的 `.c` 文件。
6. 使用语句 `extern lv_font_t my_font_name;` 声明代码，或者使用函数 `LV_FONT_DECLARE(my_font_name);`。

## 使用符号 (Using the symbol)

1. 将 Unicode 值转换为 UTF8，例如在 [站点](#) 上。对于 `0xf287`，Hex UTF-8 字节是 `EF 8A 87`。
2. 从 UTF8 值创建一个 `define` 字符串：`#define MY_USB_SYMBOL "\xEF\x8A\x87"`
3. 创建一个标签并设置文本。例如 `lv_label_set_text(label, MY_USB_SYMBOL)`

注意 - `lv_label_set_text(label, MY_USB_SYMBOL)` 在 `style.text.font` 属性中定义的字体中搜索此符号。要使用该符号，您可能需要对其进行更改。例如 `style.text.font = my_font_name`

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`

## 在运行时加载字体 (Load a font at run-time)

机翻： `lv_font_load` 可用于从文件加载字体。字体需要具有特殊的二进制格式。（不是 TTF 或 WOFF）。使用 `lv_font_conv` 和 `--format bin` 选项来生成 LVGL 兼容字体文件。

请注意，要加载字体 *LVGL* 的文件系统 需要启用，并且必须添加驱动程序。

`lv_font_load` can be used to load a font from a file. The font needs to have a special binary format. (Not TTF or WOFF). Use `lv_font_conv` with the `--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 must 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);
```

### 添加新的字体引擎 (Add a new font engine)

LVGL 的字体界面设计得非常灵活，但即便如此，同样也可以添加自己的字体引擎来代替 LVGL 的内部字体引擎。

例如，您可以使用 **FreeType** 实时渲染来自 TTF 字体的字形或使用外部闪存来存储字体的位图并在库需要时读取它们。

可以在 `lv_freetype` 存储库中找到可以使用的 FreeType。

为此，需要创建一个自定义的 `lv_font_t` 变量：

```
/*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 ...
     */
    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*/
```

(下页继续)

(续上页)

```

    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*/
}

```

### 字体退回 (Use font fallback)

您可以在 `lv_font_t` 中指定 `fallback` 以提供字体的回退。当字体找不到字母的字形时，它会尝试让字体从 `fallback` 来处理。

`fallback` can be chained, so it will try to solve until there is no `fallback` set.

```

/* Roboto font doesn't have support for CJK glyphs */
lv_font_t *roboto = my_font_load_function();
/* Droid Sans Fallback has more glyphs but its typeface doesn't look good as Roboto */
lv_font_t *droid_sans_fallback = my_font_load_function();
/* So now we can display Roboto for supported characters while having wider_
↪characters set support */
roboto->fallback = droid_sans_fallback;

```

## 1.4.12 图像 (Images)

图像在系统中是存储位图数据和元数据的文件或变量

### 储存图像 (Store images)

图像可以存储在两个地方

- 内部存储器 (RAM 或 ROM) 中的变量里
- 文件

## 变量 (Variables)

存储在变量中的图像主要由具有以下字段的 `lv_img_dsc_t` 结构体组成：

- **header**
  - *cf* 颜色格式 (Color format) . See below
  - *w* 宽度 (像素值) width in pixels ( $\leq 2048$ )
  - *h* 高度 (像素值) height in pixels ( $\leq 2048$ )
  - *always zero* 3 bits 0 值
  - *reserved* 系统保留字段
- **data** 指向存储图像本身的数组的指针 pointer to an array where the image itself is stored
- **data\_size** data 字节长度 (bytes)

这些通常放在 C 文件中并存储在项目里。它们像其他常量数据一样链接到生成的可执行文件中。

## 文件 (Files)

要处理文件，您需要向 LVGL 添加一个存储 *Drive*。简而言之，*Drive* 是在 LVGL 中注册以进行文件操作的函数 (*open*、*read*、*close* 等) 的集合。在任何情况下，*Drive* 都只是读取和/或将数据写入内存的抽象。

您可以向标准文件系统 (SD 卡上的 FAT32) 添加接口，或者创建简单的文件系统以从 SPI 闪存读取数据，在任何情况下，*Drive* 都只是读取和/或将数据写入内存的抽象。

具体请看 [File system](#)

存储为文件的图像不会链接到生成的可执行文件中，必须在显示之前读入 RAM。因此，它们不像在编译时链接的图像资源那样友好。但是，它们更容易替换而无需重建主程序。

## 颜色格式 (Color formats)

支持各种内置颜色格式：

- **LV\_IMG\_CF\_TRUE\_COLOR** 简单地存储 RGB 颜色 (以 LVGL 配置的任何颜色深度)。
- **LV\_IMG\_CF\_TRUE\_COLOR\_ALPHA** 类似于 **LV\_IMG\_CF\_TRUE\_COLOR**，但它还为每个像素添加了一个 alpha (透明度) 字节。
- **LV\_IMG\_CF\_TRUE\_COLOR\_CHROMA\_KEYED** 类似于 **LV\_IMG\_CF\_TRUE\_COLOR**，但如果像素具有 **LV\_COLOR\_TRANSP** 颜色 (在 *lv\_conf.h* 中设置)，它将是透明的。
- **LV\_IMG\_CF\_INDEXED\_1/2/4/8BIT** 使用具有 2、4、16 或 256 种颜色的调色板，并以 1、2、4 或 8 位存储每个像素。
- **LV\_IMG\_CF\_ALPHA\_1/2/4/8BIT** 仅存储 1、2、4 或 8 位的 Alpha 值像素采用 `style.img_recolor` 的颜色和设置的不透明度。源图像必须是 Alpha 通道。这非常适用于类似于字体的位图，其中整个图像是一种可以更改的颜色。

**LV\_IMG\_CF\_TRUE\_COLOR** 图像的字节按以下顺序存储

对于 32 位色深：

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



- Byte 3: Alpha

对于 16 位色深：

- 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)

对于 8 位色深：

- Byte 0: Red 3 bit, Green 3 bit, Blue 2 bit
- Byte 2: Alpha byte (only with LV\_IMG\_CF\_TRUE\_COLOR\_ALPHA)

您可以以 *Raw* 格式存储图像，以表明它没有使用其中一种内置颜色格式进行编码，并且需要使用外部图像解码器来解码图像。

- **LV\_IMG\_CF\_RAW** Indicates a basic raw image (e.g. a PNG or JPG image).
- **LV\_IMG\_CF\_RAW\_ALPHA** Indicates that an image has alpha and an alpha byte is added for every pixel.
- **LV\_IMG\_CF\_RAW\_CHROMA\_KEYED** Indicates that an image is chroma-keyed as described in **LV\_IMG\_CF\_TRUE\_COLOR\_CHROMA\_KEYED** above.

### 添加与使用图像 (Add and use images)

您可以通过两种方式将图像添加到 LVGL：

- 使用在线转换器
- 手动创建图像

#### 在线转换 (Online converter)

在线图像转换器：<https://lvgl.io/tools/imageconverter>

通过在线转换器将图像添加到 LVGL 很容易。

1. 准备 *BMP*, *PNG* or *JPG* 格式的图像。
2. 为图像指定在 LVGL 中的名称。
3. 选择颜色格式 *Color format*.
4. 选择要生成的图像类型。选择二进制文件将生成一个 **.bin** 文件，该文件必须单独存储并使用文件支持读取。选择一个变量将生成一个可以链接到您的项目的标准 C 文件。
5. 点击转换按钮。转换完成后，您的浏览器将自动下载生成的文件。

在生成的 C 数组（变量）中，所有色深（1、8、16 或 32）的位图都包含在 C 文件中，但实际上只有与 *lv\_conf.h* 中的 **LV\_COLOR\_DEPTH** 匹配的色深才会链接到生成的可执行文件中。

对于二进制文件，您需要指定所需的颜色格式：

- 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



## 手动创建 (Manually create an image)

如果您在运行时生成图像，您可以制作一个图像变量以使用 LVGL 显示它。例如：

```
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,
};
```

如果颜色格式是 LV\_IMG\_CF\_TRUE\_COLOR\_ALPHA，你可以将 data\_size 设置为 80 \* 60 \* LV\_IMG\_PX\_SIZE\_ALPHA\_BYTE。

在运行时创建和显示图像的另一个（可能更简单）方法是使用 *Canvas* 对象。

## 使用图像 (Use images)

在 LVGL 中使用图像的最简单方法是使用 *lv\_img* 对象显示它：

```
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");
```

如果图像是使用在线转换器转换的，则应在调用图像的文件使用函数 LV\_IMG\_DECLARE(my\_icon\_dsc) 来声明图像。

## 图像解码器 (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.

An image decoder consists of 4 callbacks:

- **info** get some basic info about the image (width, height and color format).
- **open** open an image: either store a decoded image or set it to **NULL** to indicate the image can be read line-by-line.
- **read** if *open* didn't fully open an image this function should give some decoded data (max 1 line) from a given position.
- **close** close an 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 decoders until it finds one which can open the image, i.e. one which knows 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.

### 自定义图像格式 (Custom image formats)

The easiest way to create a custom image is to use the online image converter and select Raw, Raw with alpha or Raw with chroma-keyed format. It will just take every byte of the binary file you uploaded and write it as an 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\_CHROMA\_KEYED accordingly. You should choose the correct format according to your needs: a fully opaque image, using an alpha channel or using a chroma key.

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 the *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 images only in *True color* format (or Raw but ultimately it will be in *True color* format). The LV\_IMG\_CF\_USER\_ENCODED... formats are not known by the library and therefore they should be decoded to one of the known formats from the *Color formats* section. It's possible to decode an 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.

### 注册图像解码器 (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` */
    ...

    header->cf = LV_IMG_CF_RAW_ALPHA;
}
```

(下页继续)

(续上页)

```

    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` */
}

/**
 * Free the allocated resources

```

(下页继续)

(续上页)

```

* @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 the entire image (e.g. no memory for it), set `dsc->img_data = NULL` and use `read_line` to get the pixel data.
- In `decoder_close` you should free all allocated resources.
- `decoder_read` is optional. Decoding the whole image requires extra memory and some computational overhead. However, it 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.

### 手动使用解码器 (Manually use an image decoder)

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

The `color` parameter is used only with `LV_IMG_CF_ALPHA_1/2/4/8BIT` images to tell color of the image. `frame_id` can be used if the image to open is an animation.

```

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

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

```

## 图像缓存 (Image caching)

有时打开图像需要很多时间。连续解码 PNG 图像或从缓慢的外部存储器加载图像将是低效的，并且用户体验不好。

因此，LVGL 可以预先缓存给定数量的图像。缓存意味着一些图像将保持打开状态，因此 LVGL 可以从通过 `dsc->img_data` 快速访问，减少加载时间。

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. Image caching may not be worth it if you have a deeply embedded target which decodes small images from a relatively fast storage medium.

## Cache size

The number of cache entries can be defined with `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 control it.)

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

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

## Memory usage

Note that a cached image might continuously consume memory. For example, if three PNG images are cached, they will consume memory while they are open.

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 from cache.

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

## API

### Image buffer

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

### 1.4.13 文件系统 (File system)

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

### Ready to use drivers

The `lv_fs_if` repository contains prepared drivers using POSIX, standard C and the [FATFS API](#). See its [README](#) for the details.

### Adding a driver

#### Registering a driver

To add a driver, a `lv_fs_drv_t` needs to be initialized like below. The `lv_fs_drv_t` needs to be static, global or dynamically allocated and not a local variable.

```
static lv_fs_drv_t drv;                                /*Needs to be static or global*/
lv_fs_drv_init(&drv);                                  /*Basic initialization*/

drv.letter = 'S';                                       /*An uppercase letter to identify the drive_
↪*/                                                    ↪
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.dir_open_cb = my_dir_open_cb;                       /*Callback to open directory to read its_
↪content */                                            ↪
```

(下页继续)

(续上页)

```

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.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 operation is not supported.

## Implementing the callbacks

### Open callback

The prototype of `open_cb` looks like this:

```
void * (*open_cb)(lv_fs_drv_t * drv, const char * path, lv_fs_mode_t mode);
```

`path` is the path after the drive letter (e.g. "S:path/to/file.txt" -> "path/to/file.txt" ). `mode` can be `LV_FS_MODE_WR` or `LV_FS_MODE_RD` to open for writes or reads.

The return value is a pointer to a *file object* that describes the opened file or `NULL` if there were any issues (e.g. the file wasn't found). The returned file object will be passed to other file system related callbacks. (see below)

### Other callbacks

The other callbacks are quite similar. For example `write_cb` looks like this:

```
lv_fs_res_t (*write_cb)(lv_fs_drv_t * drv, void * file_p, const void * buf, uint32_t_
↪btw, uint32_t * bw);
```

For `file_p`, LVGL passes the return value of `open_cb`, `buf` is the data to write, `btw` is the Bytes To Write, `bw` is the actually written bytes.

For a template of these callbacks see [lv\\_fs\\_template.c](#).

### 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 writes only 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 directories in the result but it can be a good practice to insert a '/' in front of each 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);
```

## Use drives for images

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

To use files in image widgets the following callbacks are required:

- open
- close
- read
- seek
- tell

## API

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

### 1.4.14 动画 (Animations)

You can automatically change the value of a variable between a start and an end value using animations. Animation will happen by periodically calling an “animator” function with the corresponding value parameter.

The *animator* functions have the following prototype:



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

This prototype is compatible with the majority of the property *set* functions in LVGL. For example `lv_obj_set_x(obj, value)` or `lv_obj_set_width(obj, value)`

### 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 target of the animation*/
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);

/* 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, lv_anim_path_ease_in);

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

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

/*When ready, play the animation backward with this duration. Default is 0 (disabled)
↳[ms]*/
lv_anim_set_playback_time(&a, time);

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

/*Number of repetitions. Default is 1. LV_ANIM_REPEAT_INFINITE for infinite
↳repetition*/
```

(下页继续)

(续上页)

```

lv_anim_set_repeat_count(&a, cnt);

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

/*true (default): apply the start value immediately, false: apply start value after
↳ delay when the 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 and `lv_anim_start()` will remove any existing animations for such a pair.

### Animation path

You can control the path of an animation. The most simple case is linear, meaning the current value between *start* and *end* is changed with fixed steps. A *path* is a function which calculates the next value to set based on the current state of the animation. Currently, there are the following built-in path 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
- `lv_anim_path_overshoot` overshoot the end value
- `lv_anim_path_bounce` bounce back a little from the end value (like hitting a wall)

### Speed vs time

By default, you set the animation time directly. But in some cases, setting 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 yield 5000 milliseconds. For example, in the case of `lv_obj_set_x` *unit* is pixels so 20 means 20 *px/sec* speed.

## Delete animations

You can delete an animation with `lv_anim_del(var, func)` if you provide the animated variable and its animator function.

## Timeline

A timeline is a collection of multiple animations which makes it easy to create complex composite animations.

Firstly, create an animation element but don't call `lv_anim_start()`.

Secondly, create an animation timeline object by calling `lv_anim_timeline_create()`.

Thirdly, add animation elements to the animation timeline by calling `lv_anim_timeline_add(at, start_time, &a)`. `start_time` is the start time of the animation on the timeline. Note that `start_time` will override the value of `delay`.

Finally, call `lv_anim_timeline_start(at)` to start the animation timeline.

It supports forward and backward playback of the entire animation group, using `lv_anim_timeline_set_reverse(at, reverse)`.

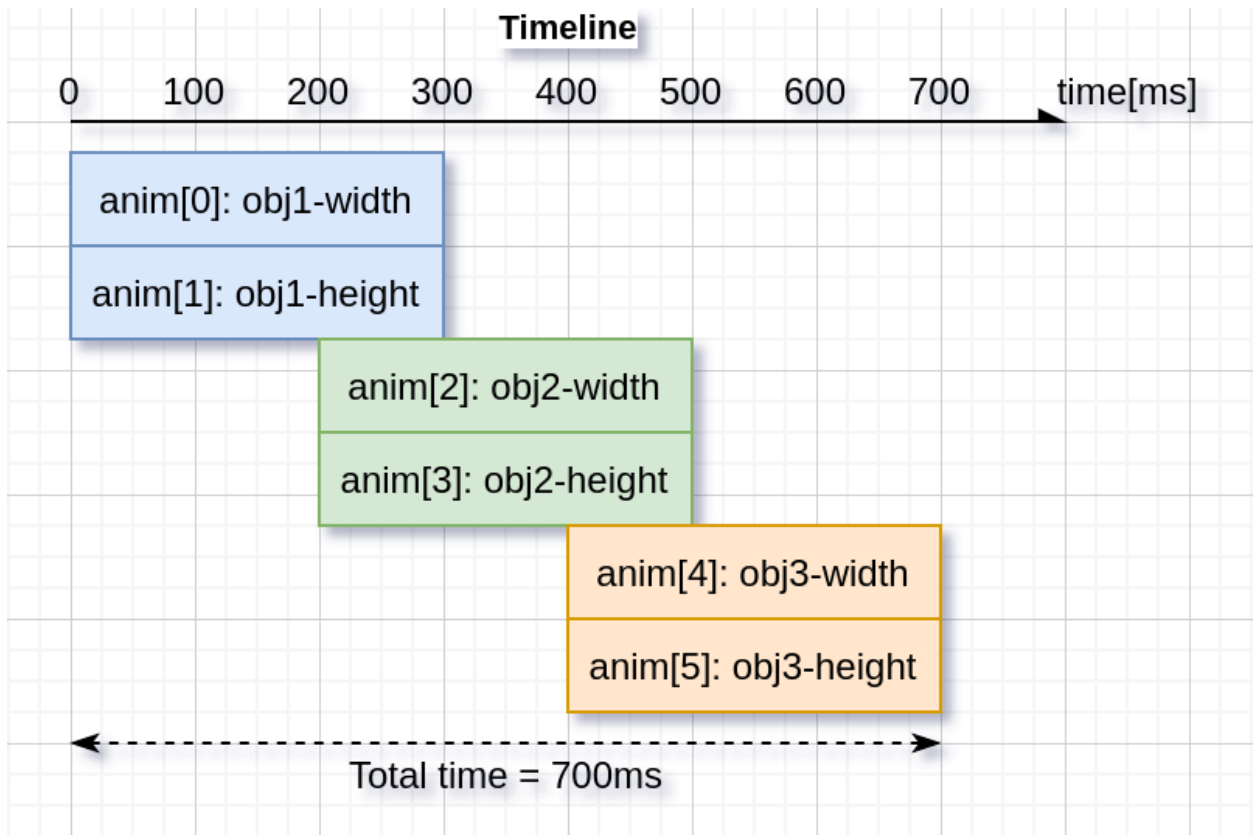
Call `lv_anim_timeline_stop(at)` to stop the animation timeline.

Call `lv_anim_timeline_set_progress(at, progress)` function to set the state of the object corresponding to the progress of the timeline.

Call `lv_anim_timeline_get_playtime(at)` function to get the total duration of the entire animation timeline.

Call `lv_anim_timeline_get_reverse(at)` function to get whether to reverse the animation timeline.

Call `lv_anim_timeline_del(at)` function to delete the animation timeline.



## Examples

## API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

### 1.4.15 定时器 (Timers)

LVGL has a built-in timer system. You can register a function to have it be called periodically. The timers are handled and called in `lv_timer_handler()`, which needs to be called every few milliseconds. See [Porting](#) for more information.

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

## Create a timer

To create a new timer, use `lv_timer_create(timer_cb, period_ms, user_data)`. It will create an `lv_timer_t *` variable, which can be used later to modify the parameters of the timer. `lv_timer_create_basic()` can also be used. This allows you to create a new timer without specifying any parameters.

A timer callback should have a `void (*lv_timer_cb_t)(lv_timer_t *)`; prototype.

For example:

```
void my_timer(lv_timer_t * timer)
{
    /*Use the user_data*/
    uint32_t * user_data = timer->user_data;
    printf("my_timer called with user data: %d\n", *user_data);

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

...

static uint32_t user_data = 10;
lv_timer_t * timer = lv_timer_create(my_timer, 500, &user_data);
```

## Ready and Reset

`lv_timer_ready(timer)` makes a timer run on the next call of `lv_timer_handler()`.

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

## Set parameters

You can modify some timer parameters later:

- `lv_timer_set_cb(timer, new_cb)`
- `lv_timer_set_period(timer, new_period)`

## Repeat count

You can make a timer repeat only a given number of times with `lv_timer_set_repeat_count(timer, count)`. The timer will automatically be deleted after it's called the defined number of times. Set the count to -1 to repeat indefinitely.

## Measure idle time

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

## Asynchronous calls

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

For example:

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

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

...

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

/*Delete screen on next call of `lv_timer_handler`, not right now.*/
lv_async_call(my_screen_cleanup, 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_cleanup` you could just use `lv_obj_del_async` which will delete the object on the next call to `lv_timer_handler`.

## API

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

### 1.4.16 Drawing

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

However, it can be useful to have a basic understanding of how drawing happens in LVGL to add customization, make it easier to find bugs or just out of curiosity.

The basic concept is to not draw directly onto the display but rather to first draw on an internal draw buffer. When a drawing (rendering) is ready that buffer is copied to the display.

The draw buffer can be smaller than a display's size. LVGL will simply render in "tiles" that fit into the given draw buffer.

This approach has two main advantages compared to directly drawing to the display:

1. It avoids flickering while the layers of the UI are drawn. For example, if LVGL drew directly onto the display, when drawing a *background + button + text*, each "stage" would be visible for a short time.
2. It's faster to modify a buffer in internal RAM and finally write one pixel only once than reading/writing the display directly on each pixel access. (e.g. via a display controller with SPI interface).

Note that this concept is different from "traditional" double buffering where there are two display sized frame buffers: one holds the current image to show on the display, and rendering happens to the other (inactive) frame buffer, and they are swapped when the rendering is finished. The main difference is that with LVGL you don't have to store two frame buffers (which usually requires external RAM) but only smaller draw buffer(s) that can easily fit into internal RAM.

## Mechanism of screen refreshing

Be sure to get familiar with the *Buffering modes of LVGL* first.

LVGL refreshes the screen in the following steps:

1. Something happens in the UI which requires redrawing. For example, a button is pressed, a chart is changed, 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 partially out of the parent are cropped to the parent's area.
  - Objects on other screens are not added.
3. In every `LV_DISP_DEF_REFR_PERIOD` (set in `lv_conf.h`) the following happens:
  - LVGL checks the invalid areas and joins those that are adjacent or intersecting.
  - Takes the first joined area, if it's smaller than the *draw buffer*, then simply renders the area's content into the *draw buffer*. If the area doesn't fit into the buffer, draw as many lines as possible to the *draw buffer*.
  - When the area is rendered, call `flush_cb` from the display driver to refresh the display.
  - If the area was larger than the buffer, render the remaining parts too.
  - Repeat the same with remaining joined areas.

When an area is redrawn the library searches the top-most object which covers that area and starts drawing 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 necessary to redraw the display under the rest of the button too.

The difference between buffering modes regarding the drawing mechanism is the following:

1. **One buffer** - LVGL needs to wait for `lv_disp_flush_ready()` (called from `flush_cb`) before starting to redraw the next part.
2. **Two 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. **Double buffering** - `flush_cb` should only swap the addresses of the frame buffers.

## Masking

*Masking* is the basic concept of LVGL's draw engine. To use LVGL it's not required to know about the mechanisms described here but you might find interesting to know how drawing works under hood. Knowing about masking comes in handy if you want to customize drawing.

To learn about masking let's see the steps of drawing first. LVGL performs the following steps to render any shape, image or text. It can be considered as a drawing pipeline.

1. **Prepare the draw descriptors** Create a draw descriptor from an object's styles (e.g. `lv_draw_rect_dsc_t`). This gives us the parameters for drawing, for example colors, widths, opacity, fonts, radius, etc.
2. **Call the draw function** Call the draw function with the draw descriptor and some other parameters (e.g. `lv_draw_rect()`). It will render the primitive shape to the current draw buffer.
3. **Create masks** If the shape is very simple and doesn't require masks, go to #5. Otherwise, create the required masks in the draw function. (e.g. a rounded rectangle mask)
4. **Calculate all the added mask** It composites opacity 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 this side should be removed.
5. **Blend a color or image** During blending, masking (make some pixels transparent or opaque), blending modes (additive, subtractive, etc.) and color/image opacity are handled.

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

- **LV\_DRAW\_MASK\_TYPE\_LINE** Removes a side from a line (top, bottom, left or right). `lv_draw_line` uses four instances of it. Essentially, every (skew) line is bounded with four line masks forming a rectangle.
- **LV\_DRAW\_MASK\_TYPE\_RADIUS** Removes the inner or outer corners of a rectangle with a radiused transition. It's also used to create circles by setting the radius to large value (`LV_RADIUS_CIRCLE`)
- **LV\_DRAW\_MASK\_TYPE\_ANGLE** Removes a circular 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 a bitmap array and the necessary parts are applied

Masks are used to create almost every basic primitive:

- **letters** Create a mask from the letter and draw a rectangle with the letter's color using the mask.
- **line** Created from four "line masks" to mask out the left, right, top and bottom part of the line to get a perfectly perpendicular perimeter.
- **rounded rectangle** A mask is created real-time to add a radius to the corners.
- **clip corner** To clip overflowing content (usually children) on rounded corners, a rounded rectangle mask is also applied.
- **rectangle border** Same as a rounded rectangle but the inner part is masked out too.
- **arc drawing** A circular border is drawn but an arc mask is applied too.
- **ARGB images** The alpha channel is separated into a mask and the image is drawn as a normal RGB image.



## Using masks

Every mask type has a related parameter structure to describe the mask's data. The following parameter types exist:

- `lv_draw_mask_line_param_t`
- `lv_draw_mask_radius_param_t`
- `lv_draw_mask_angle_param_t`
- `lv_draw_mask_fade_param_t`
- `lv_draw_mask_map_param_t`

1. Initialize a mask parameter with `lv_draw_mask_<type>_init`. See `lv_draw_mask.h` for the whole API.
2. Add the mask parameter to the draw engine with `int16_t mask_id = lv_draw_mask_add(&param, ptr)`. `ptr` can be any pointer to identify the mask, (NULL if unused).
3. Call the draw functions
4. Remove the mask from the draw engine with `lv_draw_mask_remove_id(mask_id)` or `lv_draw_mask_remove_custom(ptr)`.
5. Free the parameter with `lv_draw_mask_free_param(&param)`.

A parameter can be added and removed any number of times, but it needs to be freed when not required anymore.

`lv_draw_mask_add` saves only the pointer of the mask so the parameter needs to be valid while in use.

## Hook drawing

Although widgets can be easily customized by styles there might be cases when something more custom is required. To ensure a great level of flexibility LVGL sends a lot of events during drawing with parameters that tell what LVGL is about to draw. Some fields of these parameters can be modified to draw something else or any custom drawing operations can be added manually.

A good use case for this is the *Button matrix* widget. By default, its buttons can be styled in different states, but you can't style the buttons one by one. However, an event is sent for every button and you can, for example, tell LVGL to use different colors on a specific button or to manually draw an image on some buttons.

Each of these events is described in detail below.

## Main drawing

These events are related to the actual drawing of an object. E.g. the drawing of buttons, texts, etc. happens here.

`lv_event_get_clip_area(event)` can be used to get the current clip area. The clip area is required in draw functions to make them draw only on a limited area.

## LV\_EVENT\_DRAW\_MAIN\_BEGIN

Sent before starting to draw an object. This is a good place to add masks manually. E.g. add a line mask that “removes” the right side of an object.

## LV\_EVENT\_DRAW\_MAIN

The actual drawing of an object happens in this event. E.g. a rectangle for a button is drawn here. First, the widgets’ internal events are called to perform drawing and after that you can draw anything on top of them. For example you can add a custom text or an image.

## LV\_EVENT\_DRAW\_MAIN\_END

Called when the main drawing is finished. You can draw anything here as well and it’s also a good place to remove any masks created in LV\_EVENT\_DRAW\_MAIN\_BEGIN.

### Post drawing

Post drawing events are called when all the children of an object are drawn. For example LVGL use the post drawing phase to draw scrollbars because they should be above all of the children.

`lv_event_get_clip_area(event)` can be used to get the current clip area.

## LV\_EVENT\_DRAW\_POST\_BEGIN

Sent before starting the post draw phase. Masks can be added here too to mask out the post drawn content.

## LV\_EVENT\_DRAW\_POST

The actual drawing should happen here.

## LV\_EVENT\_DRAW\_POST\_END

Called when post drawing has finished. If masks were not removed in LV\_EVENT\_DRAW\_MAIN\_END they should be removed here.

### Part drawing

When LVGL draws a part of an object (e.g. a slider’s indicator, a table’s cell or a button matrix’s button) it sends events before and after drawing that part with some context of the drawing. This allows changing the parts on a very low level with masks, extra drawing, or changing the parameters that LVGL is planning to use for drawing.

In these events an `lv_obj_draw_part_t` structure is used to describe the context of the drawing. Not all fields are set for every part and widget. To see which fields are set for a widget refer to the widget’s documentation.

`lv_obj_draw_part_t` has the following fields:

```

// Always set
const lv_area_t * clip_area;           // The current clip area, required if you need to
↪ draw something in the event
uint32_t part;                         // The current part for which the event is sent
uint32_t id;                           // The index of the part. E.g. a button's index
↪ on button matrix or table cell index.

// Draw descriptors, set only if related
lv_draw_rect_dsc_t * rect_dsc;         // A draw descriptor that can be modified to
↪ changed what LVGL will draw. Set only for rectangle-like parts
lv_draw_label_dsc_t * label_dsc;       // A draw descriptor that can be modified to
↪ changed what LVGL will draw. Set only for text-like parts
lv_draw_line_dsc_t * line_dsc;         // A draw descriptor that can be modified to
↪ changed what LVGL will draw. Set only for line-like parts
lv_draw_img_dsc_t * img_dsc;           // A draw descriptor that can be modified to
↪ changed what LVGL will draw. Set only for image-like parts
lv_draw_arc_dsc_t * arc_dsc;           // A draw descriptor that can be modified to
↪ changed what LVGL will draw. Set only for arc-like parts

// Other parameters
lv_area_t * draw_area;                 // The area of the part being drawn
const lv_point_t * p1;                 // A point calculated during drawing. E.g. a
↪ point of a chart or the center of an arc.
const lv_point_t * p2;                 // A point calculated during drawing. E.g. a
↪ point of a chart.
char text[16];                         // A text calculated during drawing. Can be
↪ modified. E.g. tick labels on a chart axis.
lv_coord_t radius;                     // E.g. the radius of an arc (not the corner
↪ radius).
int32_t value;                         // A value calculated during drawing. E.g. Chart
↪ 's tick line value.
const void * sub_part_ptr;             // A pointer the identifies something in the part.
↪ E.g. chart series.

```

`lv_event_get_draw_part_dsc(event)` can be used to get a pointer to `lv_obj_draw_part_t`.

## LV\_EVENT\_DRAW\_PART\_BEGIN

Start the drawing of a part. This is a good place to modify the draw descriptors (e.g. `rect_dsc`), or add masks.

## LV\_EVENT\_DRAW\_PART\_END

Finish the drawing of a part. This is a good place to draw extra content on the part or remove masks added in LV\_EVENT\_DRAW\_PART\_BEGIN.

### Others

## LV\_EVENT\_COVER\_CHECK

This event is used to check whether an object fully covers an area or not.

`lv_event_get_cover_area(event)` returns a pointer to an area to check and `lv_event_set_cover_res(event, res)` can be used to set one of these results:

- LV\_COVER\_RES\_COVER the area is fully covered by the object
- LV\_COVER\_RES\_NOT\_COVER the area is not covered by the object
- LV\_COVER\_RES\_MASKED there is a mask on the object, so it does not fully cover the area

Here are some reasons why an object would be unable to fully cover an area:

- It's simply not fully in area
- It has a radius
- It doesn't have 100% background opacity
- It's an ARGB or chroma keyed image
- It does not have normal blending mode. In this case LVGL needs to know the colors under the object to apply blending properly
- It's a text, etc

In short if for any reason the area below an object is visible than the object doesn't cover that area.

Before sending this event LVGL checks if at least the widget's coordinates fully cover the area or not. If not the event is not called.

You need to check only the drawing you have added. The existing properties known by a widget are handled in its internal events. E.g. if a widget has  $> 0$  radius it might not cover an area, but you need to handle `radius` only if you will modify it and the widget won't know about it.

## LV\_EVENT\_REFR\_EXT\_DRAW\_SIZE

If you need to draw outside a widget, LVGL needs to know about it to provide extra space for drawing. Let's say you create an event which writes the current value of a slider above its knob. In this case LVGL needs to know that the slider's draw area should be larger with the size required for the text.

You can simply set the required draw area with `lv_event_set_ext_draw_size(e, size)`.

### 1.4.17 New widget

## 1.5 部件 (Widgets)

### 1.5.1 Base object (lv\_obj)

#### Overview

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

- coordinates
- parent object
- children
- contains the styles
- attributes like *Clickable*, *Scrollable*, etc.

In object-oriented thinking, it is the base class from which all other objects in LVGL are inherited.

The functions and functionalities of the 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 widget: it’s nothing more than a rectangle. In HTML terms, think of it as a `<div>`.

#### Coordinates

Only a small subset of coordinate settings is described here. To see all the features of LVGL (padding, coordinates in styles, layouts, etc) visit the [Coordinates](#) page.

#### 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)`.

#### Position

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

## Alignment

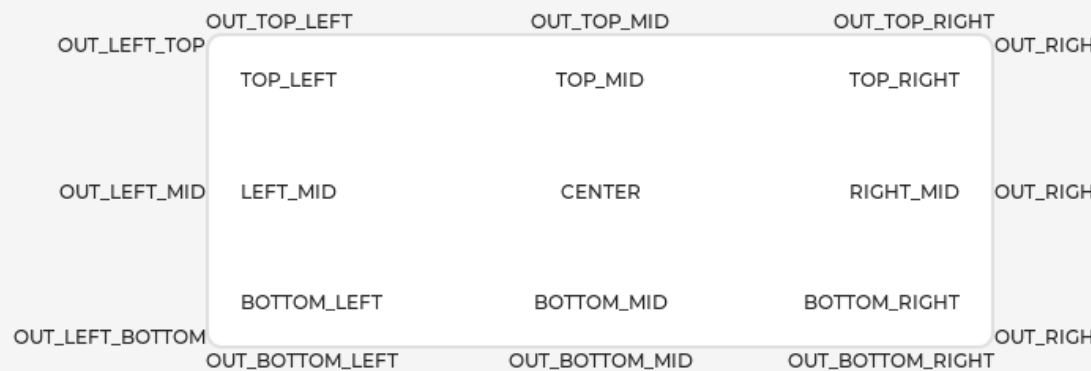
You can align the object on its parent with `lv_obj_set_align(obj, LV_ALIGN_...)`. After this every x and y setting will be relative to the set alignment mode. For example, this will shift the object by 10;20 px from the center of its parent:

```
lv_obj_set_align(obj, LV_ALIGN_CENTER);
lv_obj_set_pos(obj, 10, 20);

//Or in one function
lv_obj_align(obj, LV_ALIGN_CENTER, 10, 20);
```

To align one object to another use: `lv_obj_align_to(obj_to_align, obj_referece, LV_ALIGN_..., x, y)`

For example, to align a text below an image: `lv_obj_align_to(text, image, LV_ALIGN_OUT_BOTTOM_MID, 0, 10)`.



The following align types exist:

## 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 a specific child of a parent use `lv_obj_get_child(parent, idx)`. Some examples for `idx`:

- 0 get the child created first
- 1 get the child created second
- -1 get the child created last

The children can be iterated like this:

```
uint32_t i;
for(i = 0; i < lv_obj_get_child_cnt(parent); i++) {
    lv_obj_t * child = lv_obj_get_child(parent, i);
    /*Do something with child*/
}
```

`lv_obj_get_index(obj)` returns the index of the object in its parent. It is equivalent to the number of younger children in the parent.

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)`.

You can change the index of an object in its parent using `lv_obj_move_to_index(obj, index)`.

You can swap the position of two objects with `lv_obj_swap(obj1, obj2)`.

## Display and Screens

At the highest level of the LVGL object hierarchy is the *display* which represents the driver for a display device (physical display or simulator). A display can have one or more screens associated with it. Each screen contains a hierarchy of objects for graphical widgets representing a layout that covers the entire display.

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

If you have multiple displays, it's important to know that the screen functions operate on the most recently created display or the one explicitly selected with `lv_disp_set_default`.

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

## Events

To set an event callback for an object, use `lv_obj_add_event_cb(obj, event_cb, LV_EVENT_..., user_data)`,

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

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

## Styles

Be sure to read the [Style overview](#). Here only the most essential functions are described.

A new style can be added to an object with the `lv_obj_add_style(obj, &new_style, selector)` function. `selector` is an ORed combination of part and state(s). E.g. `LV_PART_SCROLLBAR | LV_STATE_PRESSED`.

The base objects use `LV_PART_MAIN` style properties and `LV_PART_SCROLLBAR` with the typical background style properties.

## Flags

There are some attributes which can be enabled/disabled by `lv_obj_add/clear_flag(obj, LV_OBJ_FLAG_...)`:

- `LV_OBJ_FLAG_HIDDEN` Make the object hidden. (Like it wasn't there at all)
- `LV_OBJ_FLAG_CLICKABLE` Make the object clickable by input devices
- `LV_OBJ_FLAG_CLICK_FOCUSABLE` Add focused state to the object when clicked
- `LV_OBJ_FLAG_CHECKABLE` Toggle checked state when the object is clicked
- `LV_OBJ_FLAG_SCROLLABLE` Make the object scrollable
- `LV_OBJ_FLAG_SCROLL_ELASTIC` Allow scrolling inside but with slower speed
- `LV_OBJ_FLAG_SCROLL_MOMENTUM` Make the object scroll further when "thrown"
- `LV_OBJ_FLAG_SCROLL_ONE` Allow scrolling only one snappable children
- `LV_OBJ_FLAG_SCROLL_CHAIN` Allow propagating the scroll to a parent
- `LV_OBJ_FLAG_SCROLL_ON_FOCUS` Automatically scroll object to make it visible when focused
- `LV_OBJ_FLAG_SCROLL_WITH_ARROW` Allow scrolling the focused object with arrow keys
- `LV_OBJ_FLAG_SNAPPABLE` If scroll snap is enabled on the parent it can snap to this object
- `LV_OBJ_FLAG_PRESS_LOCK` Keep the object pressed even if the press slid from the object
- `LV_OBJ_FLAG_EVENT_BUBBLE` Propagate the events to the parent too
- `LV_OBJ_FLAG_GESTURE_BUBBLE` Propagate the gestures to the parent
- `LV_OBJ_FLAG_ADV_HITTEST` Allow performing more accurate hit (click) test. E.g. accounting for rounded corners
- `LV_OBJ_FLAG_IGNORE_LAYOUT` Make the object positionable by the layouts
- `LV_OBJ_FLAG_FLOATING` Do not scroll the object when the parent scrolls and ignore layout
- `LV_OBJ_FLAG_LAYOUT_1` Custom flag, free to use by layouts
- `LV_OBJ_FLAG_LAYOUT_2` Custom flag, free to use by layouts
- `LV_OBJ_FLAG_WIDGET_1` Custom flag, free to use by widget
- `LV_OBJ_FLAG_WIDGET_2` Custom flag, free to use by widget
- `LV_OBJ_FLAG_USER_1` Custom flag, free to use by user
- `LV_OBJ_FLAG_USER_2` Custom flag, free to use by user
- `LV_OBJ_FLAG_USER_3` Custom flag, free to use by user
- `LV_OBJ_FLAG_USER_4` Custom flag, free to use by user

Some examples:

```
/*Hide on object*/
lv_obj_add_flag(obj, LV_OBJ_FLAG_HIDDEN);

/*Make an object non-clickable*/
lv_obj_clear_flag(obj, LV_OBJ_FLAG_CLICKABLE);
```



## Groups

Read the [Input devices overview](#) to learn more about *Groups*.

Objects are added to a *group* with `lv_group_add_obj(group, obj)`, and you can use `lv_obj_get_group(obj)` to see which group an object belongs to.

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

## Extended click area

By default, the objects can be clicked only within their bounding area. However, this can be extended with `lv_obj_set_ext_click_area(obj, size)`.

## Events

- `LV_EVENT_VALUE_CHANGED` when the `LV_OBJ_FLAG_CHECKABLE` flag is enabled and the object clicked (on transition to/from the checked state)
- `LV_EVENT_DRAW_PART_BEGIN` and `LV_EVENT_DRAW_PART_END` is sent for the following types:
  - `LV_OBJ_DRAW_PART_RECTANGLE` The main rectangle
    - \* `part`: `LV_PART_MAIN`
    - \* `rect_dsc`
    - \* `draw_area`: the area of the rectangle
  - `LV_OBJ_DRAW_PART_BORDER_POST` The border if the `border_post` style property is `true`
    - \* `part`: `LV_PART_MAIN`
    - \* `rect_dsc`
    - \* `draw_area`: the area of the rectangle
  - `LV_OBJ_DRAW_PART_SCROLLBAR` the scrollbars
    - \* `part`: `LV_PART_SCROLLBAR`
    - \* `rect_dsc`
    - \* `draw_area`: the area of the rectangle

Learn more about [Events](#).

## Keys

If `LV_OBJ_FLAG_CHECKABLE` is enabled, `LV_KEY_RIGHT` and `LV_KEY_UP` make the object checked, and `LV_KEY_LEFT` and `LV_KEY_DOWN` make it unchecked.

If `LV_OBJ_FLAG_SCROLLABLE` is enabled, but the object is not editable (as declared by the widget class), the arrow keys (`LV_KEY_UP`, `LV_KEY_DOWN`, `LV_KEY_LEFT`, `LV_KEY_RIGHT`) scroll the object. If the object can only scroll vertically, `LV_KEY_LEFT` and `LV_KEY_RIGHT` will scroll up/down instead, making it compatible with an encoder input device. See [Input devices overview](#) for more on encoder behaviors and the edit mode.

Learn more about [Keys](#).

## Example

## API

**警告:** doxygenfile: Unable to find project ‘lvgl’ in breathe\_projects dictionary

## 1.5.2 Core widgets

### Arc (lv\_arc)

#### Overview

The Arc consists of a background and a foreground arc. The foreground (indicator) can be touch-adjusted.

#### Parts and Styles

- **LV\_PART\_MAIN** Draws a background using the typical background style properties and an arc using the arc style properties. The arc's size and position will respect the *padding* style properties.
- **LV\_PART\_INDICATOR** Draws another arc using the *arc* style properties. Its padding values are interpreted relative to the background arc.
- **LV\_PART\_KNOB** Draws a handle on the end of the indicator using 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.

#### Usage

#### Value and range

A new value can be set using `lv_arc_set_value(arc, new_value)`. The value is interpreted in a range (minimum and maximum values) which can be modified with `lv_arc_set_range(arc, min, max)`. The default range is 0..100.

The indicator arc is drawn on the main part's arc. This if the value is set to maximum the indicator arc will cover the entire “background” arc. To set the start and end angle of the background arc use the `lv_arc_set_bg_angles(arc, start_angle, end_angle)` functions or `lv_arc_set_bg_start/end_angle(arc, angle)`.

Zero degrees is at the middle right (3 o' clock) of the object and the degrees are increasing in clockwise direction. The angles should be in the [0;360] range.

## Rotation

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

## Mode

The arc can be one of the following modes:

- `LV_ARC_MODE_NORMAL` The indicator arc is drawn from the minimum value to the current.
- `LV_ARC_MODE_REVERSE` The indicator arc is drawn counter-clockwise from the maximum value to the current.
- `LV_ARC_MODE_SYMMETRICAL` The indicator arc is drawn from the middle point to the current value.

The mode can be set by `lv_arc_set_mode(arc, LV_ARC_MODE_...)` and used only if the angle is set by `lv_arc_set_value()` or the arc is adjusted by finger.

## Change rate

If the arc is pressed the current value will set with a limited speed according to the set *change rate*. The change rate is defined in degree/second unit and can be set with `lv_arc_set_change_rate(arc, rate)`

## Setting the indicator manually

It's also possible to set the angles of the indicator arc directly with `lv_arc_set_angles(arc, start_angle, end_angle)` function or `lv_arc_set_start/end_angle(arc, start_angle)`. In this case the set "value" and "mode" are ignored.

In other words, the angle and value settings are independent. You should exclusively use one or the other. Mixing the two might result in unintended behavior.

To make the arc non-adjustable, remove the style of the knob and make the object non-clickable:

```
lv_obj_remove_style(arc, NULL, LV_PART_KNOB);  
lv_obj_clear_flag(arc, LV_OBJ_FLAG_CLICKABLE);
```

## Advanced hit test

If the `LV_OBJ_FLAG_ADV_HITTEST` flag is enabled the arc can be clicked through in the middle. Clicks are recognized only on the ring of the background arc. `lv_obj_set_ext_click_size()` makes the sensitive area larger inside and outside with the given number of pixels.

## Events

- LV\_EVENT\_VALUE\_CHANGED sent when the arc is pressed/dragged to set a new value.
- LV\_EVENT\_DRAW\_PART\_BEGIN and LV\_EVENT\_DRAW\_PART\_END are sent with the following types:
  - LV\_ARC\_DRAW\_PART\_BACKGROUND The background arc.
    - \* part: LV\_PART\_MAIN
    - \* p1: center of the arc
    - \* radius: radius of the arc
    - \* arc\_dsc
  - LV\_ARC\_DRAW\_PART\_FOREGROUND The foreground arc.
    - \* part: LV\_PART\_INDICATOR
    - \* p1: center of the arc
    - \* radius: radius of the arc
    - \* arc\_dsc
  - LV\_ARC\_DRAW\_PART\_KNOB The knob
    - \* part: LV\_PART\_KNOB
    - \* draw\_area: the area of the knob
    - \* rect\_dsc:

See the events of the *Base object* too.

Learn more about *Events*.

## Keys

- LV\_KEY\_RIGHT/UP Increases the value by one.
- LV\_KEY\_LEFT/DOWN Decreases the value by one.

Learn more about *Keys*.

## Example

## API

**警告:** doxygenfile: Unable to find project ‘lvgl’ in breathe\_projects dictionary

## Bar (lv\_bar)

### 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 the end, but also the start value of the bar can be set, which changes the start position of the indicator.

## Parts and Styles

- **LV\_PART\_MAIN** The background of the bar and it uses the typical background style properties. Adding padding makes the indicator smaller or larger. The **anim\_time** style property sets the animation time if the values set with **LV\_ANIM\_ON**.
- **LV\_PART\_INDICATOR** The indicator itself; also uses all the typical background properties.

## 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 0..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**).

## Modes

The bar can be one of the following modes:

- **LV\_BAR\_MODE\_NORMAL** A normal bar as described above
- **LV\_BAR\_SYMMETRICAL** Draw the indicator from the zero value to current value. Requires a negative minimum range and positive maximum range.
- **LV\_BAR\_RANGE** Allows setting the start value too by `lv_bar_set_start_value(bar, new_value, LV_ANIM_ON/OFF)`. The start value always has to be smaller than the end value.

## Events

- **LV\_EVENT\_DRAW\_PART\_BEGIN** and **LV\_EVENT\_DRAW\_PART\_END** are sent for the following parts:
  - **LV\_BAR\_DRAW\_PART\_INDICATOR** The indicator of the bar
    - \* **part**: **LV\_PART\_INDICATOR**
    - \* **draw\_area**: area of the indicator
    - \* **rect\_dsc**

See the events of the *Base object* too.

Learn more about *Events*.

## Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

## Example

## API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Button (lv\_btn)

### Overview

Buttons have no new features compared to the *Base object*. They are useful for semantic purposes and have slightly different default settings.

Buttons, by default, differ from Base object in the following ways:

- Not scrollable
- Added to the default group
- Default height and width set to LV\_SIZE\_CONTENT

### Parts and Styles

- LV\_PART\_MAIN The background of the button. Uses the typical background style properties.

### Usage

There are no new features compared to *Base object*.

### Events

- LV\_EVENT\_VALUE\_CHANGED when the LV\_OBJ\_FLAG\_CHECKABLE flag is enabled and the object is clicked. The event happens on transition to/from the checked state.

Learn more about [Events](#).

## Keys

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

See the events of the *Base object* too.

Learn more about *Keys*.

## Example

## API

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Button matrix (`lv_btnmatrix`)

### Overview

The Button Matrix object is a lightweight way to display multiple buttons in rows and columns. Lightweight because the buttons are not actually created but just virtually drawn on the fly. This way, one button use only eight extra bytes of memory instead of the ~100-150 bytes a normal *Button* object plus the 100 or so bytes for the the *Label* object.

The Button matrix is added to the default group (if one is set). Besides the Button matrix is an editable object to allow selecting and clicking the buttons with encoder navigation too.

### Parts and Styles

- `LV_PART_MAIN` The background of the button matrix, uses the typical background style properties. `pad_row` and `pad_column` sets the space between the buttons.
- `LV_PART_ITEMS` The buttons all use the text and typical background style properties except translations and transformations.

### 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", NULL}`. Note that the last element has to be either `NULL` or an empty string (`" "`)!

Use `"\n"` in the map to insert a **line break**. E.g. `{"btn1", "btn2", "\n", "btn3", ""}`. Each line' s buttons have their width calculated automatically. So in the example the first row will have 2 buttons each with 50% width and a second row with 1 button having 100% width.

## Control buttons

The buttons' width can be set relative to the other button in the same row 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. The width must be in the [1..7] range and the default width is 1.

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

- `LV_BTNMATRIX_CTRL_HIDDEN` Makes 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` Makes a button disabled Like `LV_STATE_DISABLED` on normal objects
- `LV_BTNMATRIX_CTRL_CHECKABLE` Enable toggling of a button. I.e. `LV_STATE_CHECKED` will be added/removed as the button is clicked
- `LV_BTNMATRIX_CTRL_CHECKED` Make the button checked. It will use the `LV_STATE_CHECKED` styles.
- `LV_BTNMATRIX_CTRL_CLICK_TRIG` Enabled: send `LV_EVENT_VALUE_CHANGE` on `CLICK`, Disabled: send `LV_EVENT_VALUE_CHANGE` on `PRESS`
- `LV_BTNMATRIX_CTRL_POPOVER` Show the button label in a popover when pressing this key
- `LV_BTNMATRIX_CTRL_RECOLOR` Enable recoloring of button texts with #. E.g. "It's #ff0000 red#"
- `LV_BTNMATRIX_CTRL_CUSTOM_1` Custom free to use flag
- `LV_BTNMATRIX_CTRL_CUSTOM_2` Custom free to use flag

By default, all flags are disabled.

To 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_BTNMATRIX_CTRL_...)` respectively. More `LV_BTNM_CTRL_...` values can be OR-ed

To set/clear the same control attribute for all buttons of a button matrix, use `lv_btnmatrix_set_btn_ctrl_all(btnm, LV_BTNM_CTRL_...)` and `lv_btnmatrix_clear_btn_ctrl_all(btnm, LV_BTNMATRIX_CTRL_...)`.

To 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_CHECKABLE`. 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 at a time.



## Events

- `LV_EVENT_VALUE_CHANGED` Sent when a button is pressed/released or repeated after long press. The event parameter is set to the ID of the pressed/released button.
- `LV_EVENT_DRAW_PART_BEGIN` and `LV_EVENT_DRAW_PART_END` are sent for the following types:
  - `LV_BTNMATRIX_DRAW_PART_BTN` The individual buttons.
    - \* `part`: `LV_PART_ITEMS`
    - \* `id`: index of the button being drawn
    - \* `draw_area`: the area of the button
    - \* `rect_dsc`

See the events of the *Base object* too.

`lv_btnmatrix_get_selected_btn(btnm)` returns the index of the most recently released or focused button or `LV_BTNMATRIX_BTN_NONE` if no such button.

`lv_btnmatrix_get_btn_text(btnm, btn_id)` returns a pointer to the text of `btn_id`th button.

Learn more about *Events*.

## Keys

- `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*.

## Example

### API

警告: doxygenfile: Unable to find project ‘lvgl’ in breathe\_projects dictionary

## Canvas (lv\_canvas)

### 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. Additionally “effects” can be applied, such as rotation, zoom and blur.

## Parts and Styles

**LV\_PART\_MAIN** Uses the typical rectangle style properties and image style properties.

## Usage

### Buffer

The Canvas needs a buffer in 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.

### Indexed colors

For `LV_IMG_CF_INDEXED_1/2/4/8` 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's color on the canvas, use `lv_canvas_set_px_color(canvas, x, y, LV_COLOR_RED)`. With `LV_IMG_CF_INDEXED_...` the index of the color needs to be passed as color. E.g. `lv_color_t c; c.full = 3;`

To set a pixel's opacity with `LV_IMG_CF_TRUE_COLOR_ALPHA` or `LV_IMG_CF_ALPHA_...` format on the canvas, use `lv_canvas_set_px_opa(canvas, x, y, opa)`.

`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_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/arc_dsc_t` variable which should be first initialized with one of `lv_draw_rect/label/img/line/arc_dsc_init()` and then 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 *draw mask* 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 another 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.

## Blur

A given area of the canvas can be blurred horizontally with `lv_canvas_blur_hor(canvas, &area, r)` or 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).

## Events

No special events are sent by canvas objects. The same events are sent as for the

See the events of the *Images* too.

Learn more about *Events*.

## Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

## Example

## API

**警告:** doxygenfile: Unable to find project ‘lvgl’ in breathe\_projects dictionary

## Checkbox (lv\_checkbox)

### Overview

The Checkbox object is created from a “tick box” and a label. When the Checkbox is clicked the tick box is toggled.

### Parts and Styles

- **LV\_PART\_MAIN** The is the background of the Checkbox and it uses the text and all the typical background style properties. `pad_column` adjusts the spacing between the tickbox and the label
- **LV\_PART\_INDICATOR** The “tick box” is a square that uses all the typical background style properties. By default, its size is equal to the height of the main part’s font. Padding properties make the tick box larger in the respective directions.

The Checkbox is added to the default group (if it is set).

### Usage

#### Text

The text can be modified with the `lv_checkbox_set_text(cb, "New text")` function and will be dynamically allocated.

To set a static text, use `lv_checkbox_set_static_text(cb, txt)`. This way, only a pointer to `txt` will be stored. The text then shouldn’t be deallocated while the checkbox exists.

### Check, uncheck, disable

You can manually check, un-check, and disable the Checkbox by using the common state add/clear function:

```
lv_obj_add_state(cb, LV_STATE_CHECKED);    /*Make the chekbox checked*/
lv_obj_clear_state(cb, LV_STATE_CHECKED); /*MAke the checkbox unchecked*/
lv_obj_add_state(cb, LV_STATE_CHECKED | LV_STATE_DISABLED); /*Make the checkbox
↪checked and disabled*/
```

## Events

- LV\_EVENT\_VALUE\_CHANGED Sent when the checkbox is toggled.
- LV\_EVENT\_DRAW\_PART\_BEGIN and LV\_EVENT\_DRAW\_PART\_END are sent for the following types:
  - LV\_CHECKBOX\_DRAW\_PART\_BOX The tickbox of the checkbox
    - \* part: LV\_PART\_INDICATOR
    - \* draw\_area: the area of the tickbox
    - \* rect\_dsc

See the events of the *Base object* too.

Learn more about *Events*.

## 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
- LV\_KEY\_ENTER Clicks the checkbox and toggles it

Note that, as usual, the state of LV\_KEY\_ENTER is translated to LV\_EVENT\_PRESSED/PRESSING/RELEASED etc.

Learn more about *Keys*.

## Example

### API

**警告:** doxygenfile: Unable to find project ‘lvgl’ in breathe\_projects dictionary

## Drop-down list (lv\_dropdown)

### 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 again.

The Drop-down list is added to the default group (if it is set). Besides the Drop-down list is an editable object to allow selecting an option with encoder navigation too.

## Parts and Styles

The Dropdown widget is built from the elements: “button” and “list” (both not related to the button and list widgets)

### Button

- **LV\_PART\_MAIN** The background of the button. Uses the typical background properties and text properties for the text on it.
- **LV\_PART\_INDICATOR** Typically an arrow symbol that can be an image or a text (**LV\_SYMBOL**).

The button goes to **LV\_STATE\_CHECKED** when it's opened.

### List

- **LV\_PART\_MAIN** The list itself. Uses the typical background properties. **max\_height** can be used to limit the height of the list.
- **LV\_PART\_SCROLLBAR** The scrollbar background, border, shadow properties and width (for its own width) and right padding for the spacing on the right.
- **LV\_PART\_SELECTED** Refers to the currently pressed, checked or pressed+checked option. Also uses the typical background properties.

The list is hidden/shown on open/close. To add styles to it use `lv_dropdown_get_list(dropdown)` to get the list object. For example:

```
lv_obj_t * list = lv_dropdown_get_list(dropdown) /*Get the list*/
lv_obj_add_style(list, &my_style, ...)          /*Add the styles to the list*/`
```

Alternatively the theme can be extended with the new styles.

## Usage

### Overview

#### Set options

Options are passed to the drop-down list as a string with `lv_dropdown_set_options(dropdown, options)`. Options should be separated by `\n`. For example: "First\nSecond\nThird". This string will be saved in the drop-down list, so it can in a local variable.

The `lv_dropdown_add_option(dropdown, "New option", pos)` function inserts a new option to `pos` index.

To save memory the options can 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

The get the *index* of the selected option, use `lv_dropdown_get_selected(dropdown)`.

`lv_dropdown_get_selected_str(dropdown, buf, buf_size)` copies the *name* of the selected option to `buf`.

## Direction

The list can be created on any side. The default `LV_DIR_BOTTOM` can be modified by `lv_dropdown_set_dir(dropdown, LV_DIR_LEFT/RIGHT/UP/BOTTOM)` function.

If the list would be vertically out of the screen, it will be aligned to the edge.

## Symbol

A symbol (typically an arrow) can be added to the dropdown list with `lv_dropdown_set_symbol(dropdown, LV_SYMBOL_...)`

If the direction of the drop-down list is `LV_DIR_LEFT` the symbol will be shown on the left, otherwise on the right.

## Show selected

The main part can either show the selected option or a static text. If a static is set with `lv_dropdown_set_text(dropdown, "Some text")` it will be shown regardless to the selected option. If the text is `NULL` the selected option is displayed on the button.

## Manually open/close

To manually open or close the drop-down list the `lv_dropdown_open/close(dropdown)` function can be used.

## Events

Apart from 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 or the list is opened/closed.
- `LV_EVENT_CANCEL` Sent when the list is closed
- `LV_EVENT_READY` Sent when the list is opened

See the events of the [Base object](#) too.

Learn more about [Events](#).

## Keys

- LV\_KEY\_RIGHT/DOWN Select the next option.
- LV\_KEY\_LEFT/UP Select the previous option.
- LV\_KEY\_ENTER Apply the selected option (Sends LV\_EVENT\_VALUE\_CHANGED event and closes the drop-down list).

Learn more about [Keys](#).

## Example

## API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## 图像部件 Image (lv\_img)

### 概览 Overview

图像是显示来自闪存（作为数组）或来自文件的图像的基本对象。图像也可以显示符号（LV\_SYMBOL\_...）。使用 [图像解码器接口 Image decoder interface](#) 也可以支持自定义图像格式。

### 部件块与样式 (Parts and Styles)

- LV\_PART\_MAIN 即显示图像的区域

### 用法 (Usage)

#### 图像源 (Image source)

为了提供最大的灵活性，图像的来源可以是：

- 代码中的变量（带有像素的 C 数组）。
- 外部存储的文件（例如在 SD 卡上）。
- 带有 [Symbols](#) 的文本。

要设置图像的来源，请使用 `lv_img_set_src(img, src)`

要从 PNG、JPG 或 BMP 图像生成像素数组，请使用 [在线图像转换工具](#) 并使用指针设置转换后的图像：  
`lv_img_set_src(img1, &converted_img_var);`

要使该变量在 C 文件中可见，您需要使用 `LV_IMG_DECLARE(converted_img_var)` 声明它。

要使用外部文件，需要使用在线转换器工具转换图像文件并选择二进制输出格式。

您还需要使用 LVGL 的文件系统模块，并为基本文件操作注册一个具有一些功能的驱动程序。转到 [文件系统](#) 了解更多信息。

要设置来自文件的图像，请使用 `lv_img_set_src(img, "S:folder1/my_img.bin")`。



您还可以设置类似于标签的符号。在这种情况下，图像将根据样式中指定的 *font* 呈现为文本。它可以使用轻量级的单色“字母”代替真实图像。你可以设置像 `lv_img_set_src(img1, LV_SYMBOL_OK)` 这样的符号。

## 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, that is the images can 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:

- **Chroma-keying** - Pixels with `LV_COLOR_CHROMA_KEY` (*lv\_conf.h*) color will be transparent.
- **Alpha byte** - An alpha byte is added to every pixel that contains the pixel's opacity

## Palette and Alpha index

Besides the *True color* (RGB) color format, the following formats are supported:

- **Indexed** - Image has a palette.
- **Alpha indexed** - Only alpha values are stored.

These options can be selected in the image converter. To learn more about the color formats, read the *Images* section.

## Recolor

A color can be mixed with every pixel of an image with a given intensity. This can be useful to show different states (checked, 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_recolor_opa` 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.

The color to mix is set by `img_recolor`.

## Auto-size

If the width or height of the image object is set to `LV_SIZE_CONTENT` the object's size will be set according to the size of the image source in the respective direction.

## Mosaic

If the object's size is greater than the image size in any directions, then the image will be repeated like a mosaic. This allows creation a large image from only a very narrow source. For example, you can have a  $300 \times 5$  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. 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.

## 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  $45.8^\circ$  set 458.

The `transform_zoom` and `transform_angle` style properties are also used to determine the final zoom and angle.

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 are higher quality but 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 not 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 objects won't change during transformation. That is `lv_obj_get_width/height/x/y()` will return the original, non-zoomed coordinates.

## Size mode

By default, when the image is zoomed or rotated the real coordinates of the image object are not changed. The larger content simply overflows the object's boundaries. It also means the layouts are not affected by the transformations.

If you need the object size to be updated to the transformed size set `lv_img_set_size_mode(img, LV_IMG_SIZE_MODE_REAL)`. (The previous mode is the default and called `LV_IMG_SIZE_MODE_VIRTUAL`). In this case if the width/height of the object is set to `LV_SIZE_CONTENT` the object's size will be set to the zoomed and rotated size. If an explicit size is set then the overflowing content will be cropped.

## Events

No special events are sent by image objects.

See the events of the *Base object* too.

Learn more about *Events*.

## Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

## Example

## API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Label (lv\_label)

### Overview

A label is the basic object type that is used to display text.

### Parts and Styles

- **LV\_PART\_MAIN** Uses all the typical background properties and the text properties. The padding values can be used to add space between the text and the background.
- **LV\_PART\_SCROLLBAR** The scrollbar that is shown when the text is larger than the widget's size.
- **LV\_PART\_SELECTED** Tells the style of the *selected text*. Only **text\_color** and **bg\_color** style properties can be used.

### Usage

#### Set text

You can set the text on a label at runtime with `lv_label_set_text(label, "New text")`. This 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. To do so, use `lv_label_set_text_static(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_text_static` (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.

## Newline

Newline characters 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 and height of the label is set to `LV_SIZE_CONTENT`. Therefore, the size of the label is automatically expanded to the text size. Otherwise, if the width or height are explicitly set (using e.g. `lv_obj_set_width` or a layout), the lines wider than the label's width can be manipulated according to several long mode policies. Similarly, the policies can be applied if the height of the text is greater than the height of the label.

- `LV_LABEL_LONG_WRAP` Wrap too long lines. If the height is `LV_SIZE_CONTENT` the label's height will be expanded, otherwise the text will be clipped. (Default)
- `LV_LABEL_LONG_DOT` Replaces the last 3 characters from bottom right corner of the label with dots (`.`)
- `LV_LABEL_LONG_SCROLL` If the text is wider than the label scroll it horizontally back and forth. If it's higher, scroll vertically. Only one direction is scrolled and horizontal scrolling has higher precedence.
- `LV_LABEL_LONG_SCROLL_CIRCULAR` If the text is wider than the label scroll it horizontally continuously. If it's higher, scroll vertically. Only one direction is scrolled and horizontal scrolling has higher precedence.
- `LV_LABEL_LONG_CLIP` Simply clip the parts of the text outside the label.

You can specify the long mode with `lv_label_set_long_mode(label, LV_LABEL_LONG_...)`

Note 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_text_static`. The buffer you pass to `lv_label_set_text_static` must be writable if you plan to use `LV_LABEL_LONG_DOT`.

## 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.

## Text selection

If enabled by `LV_LABEL_TEXT_SELECTION` part of the text can be selected. It's similar to when you use your mouse on a PC to select a text. The whole mechanism (click and select the text as you drag your finger/mouse) is implemented in *Text area* and the Label widget only allows manual text selection with `lv_label_get_text_selection_start(label, start_char_index)` and `lv_label_get_text_selection_end(label, end_char_index)`.

## Very long texts

LVGL can efficiently handle very long (e.g. > 40k characters) labels 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.

## Events

No special events are sent by the Label.

See the events of the [Base object](#) too.

Learn more about [Events](#).

## Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

## Example

## API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Line (lv\_line)

### Overview

The Line object is capable of drawing straight lines between a set of points.

### Parts and Styles

- `LV_PART_MAIN` uses all the typical background properties and line style properties.

## Usage

### Set points

The points have 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

By default, the Line's width and height are set to `LV_SIZE_CONTENT`. This means it will automatically set its size to fit all the points. If the size is set explicitly, parts on the line may not be visible.

### Invert y

By default, the  $y == 0$  point is in the top of the object. It might be counter-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 the object. *y invert* is disabled by default.

### Events

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

See the events of the *Base object* too.

Learn more about [Events](#).

### Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

### Example

### API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Roller (lv\_roller)

### Overview

Roller allows you to simply select one option from a list by scrolling.

## Parts and Styles

- **LV\_PART\_MAIN** The background of the roller uses all the typical background properties and text style properties. `style_text_line_space` adjusts the space between the options. When the Roller is scrolled and doesn't stop exactly on an option it will scroll to the nearest valid option automatically in `anim_time` milliseconds as specified in the style.
- **LV\_PART\_SELECTED** The selected option in the middle. Besides the typical background properties it uses the text style properties to change the appearance of the text in the selected area.

## Usage

### Set options

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_INFINITE` makes 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 *index* of the currently selected option use `lv_roller_get_selected(roller)`.

`lv_roller_get_selected_str(roller, buf, buf_size)` will copy the name of the selected option to `buf`.

### Visible rows

The number of visible rows can be adjusted with `lv_roller_set_visible_row_count(roller, num)`.

This function calculates the height with the current style. If the font, line space, border width, etc. of the roller changes this function needs to be called again.

## Events

- **LV\_EVENT\_VALUE\_CHANGED** Sent when a new option is selected.

See the events of the *Base object* too.

Learn more about *Events*.

## Keys

- `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)

## Example

## API

警告: doxygenfile: Unable to find project ‘lvgl’ in breathe\_projects dictionary

## Slider (`lv_slider`)

### Overview

The Slider object looks like a [Bar](#) supplemented with a knob. The knob can be dragged to set a value. Just like Bar, Slider can be vertical or horizontal.

### Parts and Styles

- `LV_PART_MAIN` The background of the slider. Uses all the typical background style properties. `padding` makes the indicator smaller in the respective direction.
- `LV_PART_INDICATOR` The indicator that shows the current state of the slider. Also uses all the typical background style properties.
- `LV_PART_KNOB` A rectangle (or circle) drawn at the current value. Also uses all the typical background properties to describe the knob(s). By default, the knob is square (with an optional corner 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.

### Usage

#### Value and range

To set an initial value use `lv_slider_set_value(slider, new_value, LV_ANIM_ON/OFF)`. The animation time is set by the styles' `anim_time` property.

To specify the range (min, max values), `lv_slider_set_range(slider, min , max)` can be used.



## Modes

The slider can be one of the following modes:

- `LV_SLIDER_MODE_NORMAL` A normal slider as described above
- `LV_SLIDER_SYMMETRICAL` Draw the indicator from the zero value to current value. Requires negative minimum range and positive maximum range.
- `LV_SLIDER_RANGE` Allows setting the start value too by `lv_bar_set_start_value(bar, new_value, LV_ANIM_ON/OFF)`. The start value has to be always smaller than the end value.

The mode can be changed with `lv_slider_set_mode(slider, LV_SLIDER_MODE_...)`

## Knob-only mode

Normally, the slider can be adjusted either by dragging the knob, or by 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 adding the `LV_OBJ_FLAG_ADV_HITTEST`: `lv_obj_add_flag(slider, LV_OBJ_FLAG_ADV_HITTEST)`.

## Events

- `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 once when released. Use `lv_slider_is_dragged` to determine whether the Slider is still being dragged or has just been released.
- `LV_EVENT_DRAW_PART_BEGIN` and `LV_EVENT_DRAW_PART_END` are sent for the following parts.
  - `LV_SLIDER_DRAW_PART_KNOB` The main (right) knob of the slider
    - \* `part`: `LV_PART_KNOB`
    - \* `draw_area`: area of the indicator
    - \* `rect_dsc`
    - \* `id`: 0
  - `LV_SLIDER_DRAW_PART_KNOB` The left knob of the slider
    - \* `part`: `LV_PART_KNOB`
    - \* `draw_area`: area of the indicator
    - \* `rect_dsc`
    - \* `id`: 1

See the events of the [Bar](#) too.

Learn more about [Events](#).

## Keys

- LV\_KEY\_UP/RIGHT Increment the slider's value by 1
- LV\_KEY\_DOWN/LEFT Decrement the slider's value by 1

Learn more about [Keys](#).

## Example

## API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Switch (lv\_switch)

### Overview

The Switch looks like a little slider and can be used to turn something on and off.

### Parts and Styles

- LV\_PART\_MAIN The background of the switch uses all the typical background style properties. **padding** makes the indicator smaller in the respective direction.
- LV\_PART\_INDICATOR The indicator that shows the current state of the switch. Also uses all the typical background style properties.
- LV\_PART\_KNOB A rectangle (or circle) drawn at left or right side of the indicator. Also uses all the typical background properties to describe the knob(s). By default the knob is square (with a optional corner 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.

### Usage

#### Change state

When the switch is turned on it goes to LV\_STATE\_CHECKED. To get the current state of the switch use `lv_obj_has_state(switch, LV_STATE_CHECKED)`. To manually turn the switch on/off call `lv_obj_add/clear_state(switch, LV_STATE_CHECKED)`.

## Events

- LV\_EVENT\_VALUE\_CHANGED Sent when the switch changes state.

See the events of the *Base object* too.

Learn more about *Events*.

## Keys

- LV\_KEY\_UP/RIGHT Turns on the slider
- LV\_KEY\_DOWN/LEFT Turns off the slider
- LV\_KEY\_ENTER Toggles the switch

Learn more about *Keys*.

## Example

## API

**警告:** doxygenfile: Unable to find project ‘lvgl’ in breathe\_projects dictionary

## Table (lv\_table)

### Overview

Tables, as usual, are built from rows, columns, and cells containing texts.

The Table object is very lightweight because only the texts are stored. No real objects are created for cells but they are just drawn on the fly.

The Table is added to the default group (if it is set). Besides the Table is an editable object to allow selecting a cell with encoder navigation too.

### Parts and Styles

- LV\_PART\_MAIN The background of the table uses all the typical background style properties.
- LV\_PART\_ITEMS The cells of the table also use all the typical background style properties and the text properties.

## Usage

### Set cell value

The cells can store only text so numbers need 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 breaks can be used in the text like `"Value\n60.3"`.

New rows and columns are automatically added is required

### Rows and Columns

To explicitly 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.

### Merge cells

Cells can be merged horizontally with `lv_table_add_cell_ctrl(table, row, col, LV_TABLE_CELL_CTRL_MERGE_RIGHT)`. To merge more adjacent cells call this function for each cell.

### Scroll

If the label's width or height is set to `LV_SIZE_CONTENT` that size will be used to show the whole table in the respective direction. E.g. `lv_obj_set_size(table, LV_SIZE_CONTENT, LV_SIZE_CONTENT)` automatically sets the table size to show all the columns and rows.

If the width or height is set to a smaller number than the “intrinsic” size then the table becomes scrollable.

### Events

- `LV_EVENT_VALUE_CHANGED` Sent when a new cell is selected with keys.
- `LV_EVENT_DRAW_PART_BEGIN` and `LV_EVENT_DRAW_PART_END` are sent for the following types:
  - `LV_TABLE_DRAW_PART_CELL` The individual cells of the table
    - \* `part`: `LV_PART_ITEMS`
    - \* `draw_area`: area of the indicator
    - \* `rect_dsc`
    - \* `label_dsc`

\* `id`: current row  $\times$  col count + current column

See the events of the *Base object* too.

Learn more about *Events*.

## Keys

The following *Keys* are processed by the Tables:

- `LV_KEY_RIGHT/LEFT/UP/DOWN/` Select a cell.

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

`lv_table_get_selected_cell(table, &row, &col)` can be used to get the currently selected cell. Row and column will be set to `LV_TABLE_CELL_NONE` if no cell is selected.

Learn more about *Keys*.

## Example

### MicroPython

No examples yet.

## API

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Text area (`lv_textarea`)

### Overview

The Text Area is a *Base object* 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.

One line mode and password modes are supported.

### Parts and Styles

- `LV_PART_MAIN` The background of the text area. Uses all the typical background style properties and the text related style properties including `text_align` to align the text to the left, right or center.
- `LV_PART_SCROLLBAR` The scrollbar that is shown when the text is too long.
- `LV_PART_SELECTED` Determines the style of the *selected text*. Only `text_color` and `bg_color` style properties can be used. `bg_color` should be set directly on the label of the text area.

- `LV_PART_CURSOR` Marks the position where the characters are inserted. 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 opacity to `LV_PART_CURSOR`'s style. The create line cursor leave the cursor transparent and set a left border. The `anim_time` style property sets the cursor's blink time.
- `LV_PART_TEXTAREA_PLACEHOLDER` Unique to Text Area, allows styling the placeholder text.

## 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 like `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 applied the cursor will jump to the position where the Text area was clicked.

## Hide the cursor

The cursor is always visible, however it can be a good idea to style it to be visible only in `LV_STATE_FOCUSED` state.

## One line mode

The Text area can be configured to be on a single line with `lv_textarea_set_one_line(textarea, true)`. In this mode the height is set automatically to show only one line, line break characters are ignored, and word wrap is disabled.

## Password mode

The text area supports password mode which can be enabled with `lv_textarea_set_password_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)` returns the actual text entered, not the bullet characters.

The visibility time can be adjusted with `LV_TEXTAREA_DEF_PWD_SHOW_TIME` in `lv_conf.h`.

## Accepted characters

You can set a list of accepted characters with `lv_textarea_set_accepted_chars(textarea, "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), scrolling and drawing might be slow. However, by enabling `LV_LABEL_LONG_TXT_HINT 1` in `lv_conf.h` the performance can be hugely improved. This will save some additional information 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

Any part of the text can be selected if enabled with `lv_textarea_set_text_selection(textarea, true)`. This works much like when you select text on your PC with your mouse.

## Events

- **LV\_EVENT\_INSERT** Sent right before a character or text is inserted. The event parameter is the text about to be inserted. `lv_textarea_set_insert_replace(textarea, "New text")` replaces the text to insert. The new text cannot be in a local variable which is destroyed when the event callback exists. "" means do not insert anything.
- **LV\_EVENT\_VALUE\_CHANGED** Sent when the content of the text area has been changed.
- **LV\_EVENT\_READY** Sent when **LV\_KEY\_ENTER** is pressed (or sent) to a one line text area.

See the events of the *Base object* too.

Learn more about *Events*.

## Keys

- **LV\_KEY\_UP/DOWN/LEFT/RIGHT** Move the cursor
- **Any character** Add the character to the current cursor position

Learn more about *Keys*.

## Example

## API

**警告:** doxygenfile: Unable to find project ‘lvgl’ in breathe\_projects dictionary

## 1.5.3 Extra widgets

### Animation Image (lv\_animimg)

#### Overview

The animation image is similar to the normal ‘Image’ object. The only difference is that instead of one source image, you set an array of multiple source images.

You can specify a duration and repeat count.

#### Parts and Styles

- **LV\_PART\_MAIN** A background rectangle that uses the typical background style properties and the image itself using the image style properties.



## Usage

### Image sources

To set the image in a state, use the `lv_animimg_set_src(imgbtn, dsc[], num)`.

### Events

No special events are sent by image objects.

See the events of the Base object too.

Learn more about [Events](#).

### Keys

No Keys are processed by the object type.

Learn more about [Keys](#).

### Example

### API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Calendar (lv\_calendar)

### Overview

The Calendar object is a classic calendar which can:

- show the days of any month in a 7x7 matrix
- Show the name of the days
- highlight the current day (today)
- highlight any user-defined dates

The Calendar is added to the default group (if it is set). Calendar is an editable object which allow selecting and clicking the dates with encoder navigation too.

To make the Calendar flexible, by default it doesn't show the current year or month. Instead, there are optional "headers" that can be attached to the calendar.

## Parts and Styles

The calendar object uses the *Button matrix* object under the hood to arrange the days into a matrix.

- **LV\_PART\_MAIN** The background of the calendar. Uses all the background related style properties.
- **LV\_PART\_ITEMS** Refers to the dates and day names. Button matrix control flags are set to differentiate the buttons and a custom drawer event is added modify the properties of the buttons as follows:
  - day names have no border, no background and drawn with a gray color
  - days of the previous and next month have **LV\_BTNMATRIX\_CTRL\_DISABLED** flag
  - today has a thicker border with the theme's primary color
  - highlighted days have some opacity with the theme's primary color.

## Usage

Some functions use the `lv_calendar_date_t` type 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, year, month, day)` function. `month` needs to be in 1..12 range and `day` in 1..31 range.

### Shown date

To set the shown date, use `lv_calendar_set_shown_date(calendar, year, month);`

### 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, date_num)`. 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", ...}`; Only the pointer of the day names is saved so the elements should be static, global or constant variables.

## Events

- `LV_EVENT_VALUE_CHANGED` Sent if a date is clicked. `lv_calendar_get_pressed_date(calendar, &date)` set `date` to the date currently being pressed. Returns `LV_RES_OK` if there is a valid pressed date, else `LV_RES_INV`.

Learn more about [Events](#).

## Keys

- `LV_KEY_RIGHT/UP/LEFT/RIGHT` To navigate among the buttons to dates
- `LV_KEY_ENTER` To press/release the selected date

Learn more about [Keys](#).

## Headers

**From v8.1 the header is added directly into the Calendar widget and the API of the headers has been changed.**

### Arrow buttons

`lv_calendar_header_arrow_create(calendar)` creates a header that contains a left and right arrow on the sides and a text with the current year and month between them.

### Drop-down

`lv_calendar_header_dropdown_create(calendar)` creates a header that contains 2 drop-down lists: one for the year and another for the month.

## Example

### API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Chart (lv\_chart)

### Overview

Charts are a basic object to visualize data points. Currently *Line* charts (connect points with lines and/or draw points on them) and *Bar* charts are supported.

Charts can have:

- division lines
- 2 y axis

- axis ticks and texts on ticks
- cursors
- scrolling and zooming

## Parts and Styles

- **LV\_PART\_MAIN** The background of the chart. Uses all the typical background and *line* (for the division lines) related style properties. *Padding* makes the series area smaller.
- **LV\_PART\_SCROLLBAR** The scrollbar used if the chart is zoomed. See the *Base object*'s documentation for details.
- **LV\_PART\_ITEMS** Refers to the line or bar series.
  - Line chart: The *line* properties are used by the lines. **width**, **height**, **bg\_color** and **radius** is used to set the appearance of points.
  - Bar chart: The typical background properties are used to style the bars.
- **LV\_PART\_INDICATOR** Refers to the points on line and scatter chart (small circles or squares).
- **LV\_PART\_CURSOR** *Line* properties are used to style the cursors. **width**, **height**, **bg\_color** and **radius** are used to set the appearance of points.
- **LV\_PART\_TICKS** *Line* and *Text* style properties are used to style the ticks

## Usage

### Chart type

The following data display types exist:

- **LV\_CHART\_TYPE\_NONE** Do not display any data. Can be used to hide the series.
- **LV\_CHART\_TYPE\_LINE** Draw lines between the data points and/or points (rectangles or circles) on the data points.
- **LV\_CHART\_TYPE\_BAR** - Draw bars.
- **LV\_CHART\_TYPE\_SCATTER** - X/Y chart drawing point's and lines between the points. .

You can specify the display type with `lv_chart_set_type(chart, LV_CHART_TYPE_...)`.

### Data series

You can add any number of series to the charts by `lv_chart_add_series(chart, color, axis)`. This allocates an `lv_chart_series_t` structure which contains the chosen **color** and an array for the data points. **axis** can have the following values:

- **LV\_CHART\_AXIS\_PRIMARY\_Y** Left axis
- **LV\_CHART\_AXIS\_SECONDARY\_Y** Right axis
- **LV\_CHART\_AXIS\_PRIMARY\_X** Bottom axis
- **LV\_CHART\_AXIS\_SECONDARY\_X** Top axis

`axis` tells which axis' s range should be used to scale the values.

`lv_chart_set_ext_y_array(chart, ser, value_array)` makes the chart use an external array for the given series. `value_array` should look like this: `lv_coord_t * value_array[num_points]`. The array size needs to be large enough to hold all the points of that series. The array' s pointer will be saved in the chart so it needs to be global, static or dynamically allocated. Note: you should call `lv_chart_refresh(chart)` after the external data source has been updated to update the chart.

The value array of a series can be obtained with `lv_chart_get_y_array(chart, ser)`, which can be used with `ext_array` or *normal arrays*.

For `LV_CHART_TYPE_SCATTER` type `lv_chart_set_ext_x_array(chart, ser, value_array)` and `lv_chart_get_x_array(chart, ser)` can be used as well.

## 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_value_by_id(chart, ser, id, value)` where `id` is the index of the point you wish to update.
3. Use the `lv_chart_set_next_value(chart, ser, value)`.
4. Initialize all points to a given value with: `lv_chart_set_all_value(chart, ser, value)`.

Use `LV_CHART_POINT_NONE` as value to make the library skip drawing that point, column, or line segment.

For `LV_CHART_TYPE_SCATTER` type `lv_chart_set_value_by_id2(chart, ser, id, value)` and `lv_chart_set_next_value2(chart, ser, x_value, y_value)` can be used as well.

## Update modes

`lv_chart_set_next_value` 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 to the right.
- `LV_CHART_UPDATE_MODE_CIRCULAR` - Add the new data in circular fashion, 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, so you need to be sure the external array is large enough.

## Handling large number of points

On line charts, if the number of points is greater than the pixels horizontally, the Chart will draw only vertical lines to make the drawing of large amount of data effective. If there are, let's say, 10 points to a pixel, LVGL searches the smallest and the largest value and draws a vertical lines between them to ensure no peaks are missed.

## Vertical range

You can specify the minimum and maximum values in y-direction with `lv_chart_set_range(chart, axis, min, max)`. `axis` can be `LV_CHART_AXIS_PRIMARY` (left axis) or `LV_CHART_AXIS_SECONDARY` (right axis).

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. If there is a visible border on a side and no padding on that side, the division line would be drawn on top of the border and therefore it won't be drawn.

## Override default start point for series

If you want 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.

Note that `LV_CHART_UPDATE_MODE_SHIFT` also changes the `start_point`.

## Tick marks and labels

Ticks and labels can be added to the axis with `lv_chart_set_axis_tick(chart, axis, major_len, minor_len, major_cnt, minor_cnt, label_en, draw_size)`.

- `axis` can be `LV_CHART_AXIS_X/PRIMARY_Y/SECONDARY_Y`
- `major_len` is the length of major ticks
- `minor_len` is the length of minor ticks
- `major_cnt` is the number of major ticks on the axis
- `minor_cnt` in the number of minor ticks between two major ticks
- `label_en true`: enable label drawing on major ticks
- `draw_size` extra size required to draw the tick and labels (start with 20 px and increase if the ticks/labels are clipped)

## Zoom

The chart can be zoomed independently in x and y directions with `lv_chart_set_zoom_x(chart, factor)` and `lv_chart_set_zoom_y(chart, factor)`. If `factor` is 256 there is no zoom. 512 means double zoom, etc. Fractional values are also possible but < 256 value is not allowed.

## Cursor

A cursor can be added with `lv_chart_cursor_t * c1 = lv_chart_add_cursor(chart, color, dir);`. The possible values of `dir` `LV_DIR_NONE/RIGHT/UP/LEFT/DOWN/HOR/VER/ALL` or their OR-ed values to tell in which direction(s) should the cursor be drawn.

`lv_chart_set_cursor_pos(chart, cursor, &point)` sets the position of the cursor. `pos` is a pointer to an `lv_point_t` variable. E.g. `lv_point_t point = {10, 20};`. If the chart is scrolled the cursor will remain in the same place.

`lv_chart_get_point_pos_by_id(chart, series, id, &point_out)` gets the coordinate of a given point. It's useful to place the cursor at a given point.

`lv_chart_set_cursor_point(chart, cursor, series, point_id)` sticks the cursor at a point. If the point's position changes (new value or scrolling) the cursor will move with the point.

## Events

- `LV_EVENT_VALUE_CHANGED` Sent when a new point is clicked pressed. `lv_chart_get_pressed_point(chart)` returns the zero-based index of the pressed point.
- `LV_EVENT_DRAW_PART_BEGIN` and `LV_EVENT_DRAW_PART_END` are sent with the following types:
  - `LV_CHART_DRAW_PART_DIV_LINE_INIT` Used before/after drawn the div lines to add masks to any extra drawings. The following fields are set:
    - \* `part`: `LV_PART_MAIN`
    - \* `line_dsc`
  - `LV_CHART_DRAW_PART_DIV_LINE_HOR`, `LV_CHART_DRAW_PART_DIV_LINE_VER` Used for each horizontal and vertical division lines.
    - \* `part`: `LV_PART_MAIN`
    - \* `id`: index of the line
    - \* `p1, p2`: points of the line
    - \* `line_dsc`
  - `LV_CHART_DRAW_PART_LINE_AND_POINT` Used on line and scatter charts for lines and points.
    - \* `part`: `LV_PART_ITEMS`
    - \* `id`: index of the point
    - \* `value`: value of `id`th point
    - \* `p1, p2`: points of the line
    - \* `draw_area`: area of the point
    - \* `line_dsc`

- \* rect\_dsc
- \* sub\_part\_ptr: pointer to the series
- LV\_CHART\_DRAW\_PART\_BAR Used on bar charts for the rectangles.
  - \* part: LV\_PART\_ITEMS
  - \* id: index of the point
  - \* value: value of idth point
  - \* draw\_area: area of the point
  - \* rect\_dsc:
  - \* sub\_part\_ptr: pointer to the series
- LV\_CHART\_DRAW\_PART\_CURSOR Used on cursor lines and points.
  - \* part: LV\_PART\_CURSOR
  - \* p1, p2: points of the line
  - \* line\_dsc
  - \* rect\_dsc
  - \* draw\_area: area of the points
- LV\_CHART\_DRAW\_PART\_TICK\_LABEL Used on tick lines and labels.
  - \* part: LV\_PART\_TICKS
  - \* id: axis
  - \* value: value of the tick
  - \* text: value converted to decimal or NULL for minor ticks
  - \* line\_dsc,
  - \* label\_dsc,

See the events of the *Base object* too.

Learn more about *Events*.

## Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

## Example

## API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary



## Color wheel (lv\_colorwheel)

### Overview

As its name implies *Color wheel* allows the user to select a color. The Hue, Saturation and Value of the color can be selected separately.

Long pressing the object, the color wheel will change to the next parameter of the color (hue, saturation or value). A double click will reset the current parameter.

### Parts and Styles

- LV\_PART\_MAIN Only `arc_width` is used to set the width of the color wheel
- LV\_PART\_KNOB A 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 arc.

### Usage

#### Create a color wheel

`lv_colorwheel_create(parent, knob_recolor)` creates a new color wheel. With `knob_recolor=true` the knob's background color will be set to the current color.

#### Set color

The color can be set manually with `lv_colorwheel_set_hue/saturation/value(colorwheel, x)` or all at once with `lv_colorwheel_set_hsv(colorwheel, hsv)` or `lv_colorwheel_set_color(colorwheel, rgb)`

#### Color mode

The current color mode can be manually selected with `lv_colorwheel_set_mode(colorwheel, LV_COLORWHEEL_MODE_HUE/SATURATION/VALUE)`.

The color mode can be fixed (so as to not change with long press) using `lv_colorwheel_set_mode_fixed(colorwheel, true)`

### Events

- LV\_EVENT\_VALUE\_CHANGED Sent if a new color is selected.

Learn more about [Events](#).

## 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 value by 1
- LV\_KEY\_ENTER A long press will show the next mode. Double click to reset the current parameter.

Learn more about [Keys](#).

## Example

## API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Image button (lv\_imgbtn)

### 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.

You can set a left, right and center image, and the center image will be repeated to match the width of the object.

### Parts and Styles

- LV\_PART\_MAIN Refers to the image(s). If background style properties are used, a rectangle will be drawn behind the image button.

### Usage

### Image sources

To set the image in a state, use the `lv_imgbtn_set_src(imgbtn, LV_IMGBTN_STATE_..., src_left, src_center, src_right)`.

The image sources work the same as described in the [Image object](#) except that "Symbols" are not supported by the Image button. Any of the sources can NULL.

The possible states are:

- LV\_IMGBTN\_STATE\_RELEASED
- LV\_IMGBTN\_STATE\_PRESSED
- LV\_IMGBTN\_STATE\_DISABLED
- LV\_IMGBTN\_STATE\_CHECKED\_RELEASED
- LV\_IMGBTN\_STATE\_CHECKED\_PRESSED
- LV\_IMGBTN\_STATE\_CHECKED\_DISABLED

If you set sources only in `LV_IMGBTN_STATE_RELEASED`, these sources will be used in other states too. If you set e.g. `LV_IMGBTN_STATE_PRESSED` they will be used in pressed state instead of the released images.

## States

Instead of the regular `lv_obj_add/clear_state()` functions the `lv_imgbtn_set_state(imgbtn, LV_IMGBTN_STATE_...)` functions should be used to manually set a state.

## Events

- `LV_EVENT_VALUE_CHANGED` Sent when the button is toggled.

Learn more about [Events](#).

## Keys

- `LV_KEY_RIGHT/UP` Go to toggled state if `LV_OBJ_FLAG_CHECKABLE` is enabled.
- `LV_KEY_LEFT/DOWN` Go to non-toggled state if `LV_OBJ_FLAG_CHECKABLE` is enabled.
- `LV_KEY_ENTER` Clicks the button

Learn more about [Keys](#).

## Example

## API

**警告:** doxygenfile: Unable to find project ‘lvgl’ in breathe\_projects dictionary

## Keyboard (lv\_keyboard)

### Overview

The Keyboard object is a special *Button matrix* with predefined keymaps and other features to realize a virtual keyboard to write texts into a *Text area*.

### Parts and Styles

Similarly to Button matrices Keyboards consist of 2 part:

- `LV_PART_MAIN` The main part. Uses all the typical background properties
- `LV_PART_ITEMS` The buttons. Also uses all typical background properties as well as the *text* properties.

## 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_NUMBER` Display numbers, +/- sign, and decimal dot
- `LV_KEYBOARD_MODE_USER_1` through `LV_KEYBOARD_MODE_USER_4` User-defined modes.

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)`.

### Key Popovers

To enable key popovers on press, like on common Android and iOS keyboards, use `lv_keyboard_set_popovers(kb, true)`. The default control maps are preconfigured to only show the popovers on keys that produce a symbol and not on e.g. space. If you use a custom keymap, set the `LV_BTNMATRIX_CTRL_POPOVER` flag for all keys that you want to show a popover.

Note that popovers for keys in the top row will draw outside the widget boundaries. To account for this, reserve extra free space on top of the keyboard or ensure that the keyboard is added *after* any widgets adjacent to its top boundary so that the popovers can draw over those.

The popovers currently are merely a visual effect and don't allow selecting additional characters such as accents yet.

### 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` or `LV_SYMBOL_KEYBOARD` Close.
- `LV_SYMBOL_BACKSPACE` Delete on the left.
- `LV_SYMBOL_LEFT` Move the cursor left.
- `LV_SYMBOL_RIGHT` Move the cursor right.
- `LV_SYMBOL_NEW_LINE` New line.
- `"ABC"` Load the uppercase map.

- “*abc*” Load the lower case map.
- “*I#*” Load the lower case map.

## Events

- `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_READY` - 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`, which handles the button pressing, map changing, the assigned text area, etc. You can remove it and replace it with a custom event handler if you wish.

---

**注解:** In 8.0 and newer, adding an event handler to the keyboard does not remove the default event handler. This behavior differs from v7, where adding an event handler would always replace the previous one.

---

Learn more about [Events](#).

## Keys

- `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](#).

## Examples

## API

**警告:** doxygenfile: Unable to find project ‘lvgl’ in breathe\_projects dictionary

## LED (`lv_led`)

### Overview

The LEDs are rectangle-like (or circle) object whose brightness can be adjusted. With lower brightness the colors of the LED become darker.

## Parts and Styles

The LEDs have only one main part, called `LV_LED_PART_MAIN` and it uses all the typical background style properties.

## Usage

### Color

You can set the color of the LED with `lv_led_set_color(led, lv_color_hex(0xff0080))`. This will be used as background color, border color, and shadow color.

### 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.

### Events

- `LV_EVENT_DRAW_PART_BEGIN` and `LV_EVENT_DRAW_PART_END` is sent for the following types:
  - `LV_LED_DRAW_PART_RECTANGLE` The main rectangle. `LV_OBJ_DRAW_PART_RECTANGLE` is not sent by the base object.
    - \* `part`: `LV_PART_MAIN`
    - \* `rect_dsc`
    - \* `draw_area`: the area of the rectangle

See the events of the *Base object* too.

Learn more about *Events*.

### Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

## Example

## API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## List (lv\_list)

### Overview

The List is basically a rectangle with vertical layout to which Buttons and Texts can be added

### Parts and Styles

#### Background

- LV\_PART\_MAIN The main part of the list that uses all the typical background properties
- LV\_PART\_SCROLLBAR The scrollbar. See the [Base objects](#) documentation for details.

**Buttons and Texts** See the [Button](#)'s and [Label](#)'s documentation.

### Usage

#### Buttons

`lv_list_add_btn(list, icon, text)` adds a full-width button with an icon - that can be an image or symbol - and a text.

The text starts to scroll horizontally if it's too long.

#### Texts

`lv_list_add_text(list, icon, text)` adds a text.

#### Events

No special events are sent by the List, but sent by the Button as usual.

Learn more about [Events](#).

## Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

## Example

## API

**警告:** doxygenfile: Unable to find project ‘lvgl’ in breathe\_projects dictionary

## Menu (lv\_menu)

### Overview

The menu widget can be used to easily create multi-level menus. It handles the traversal between pages automatically.

### Parts and Styles

The menu widget is built from the following objects:

- Main container: lv\_menu\_main\_cont
  - Main header: lv\_menu\_main\_header\_cont
    - \* Back btn: [lv\\_btn](#)
      - Back btn icon: [lv\\_img](#)
  - Main page: lv\_menu\_page
- Sidebar container: lv\_menu\_sidebar\_cont
  - Sidebar header: lv\_menu\_sidebar\_header\_cont
    - \* Back btn: [lv\\_btn](#)
      - Back btn icon: [lv\\_img](#)
  - Sidebar page: lv\_menu\_page

### Usage

#### Create a menu

`lv_menu_create(parent)` creates a new empty menu.



## Header mode

The following header modes exist:

- `LV_MENU_HEADER_TOP_FIXED` Header is positioned at the top.
- `LV_MENU_HEADER_TOP_UNFIXED` Header is positioned at the top and can be scrolled out of view.
- `LV_MENU_HEADER_BOTTOM_FIXED` Header is positioned at the bottom.

You can set header modes with `lv_menu_set_mode_header(menu, LV_MENU_HEADER...)`.

## Root back button mode

The following root back button modes exist:

- `LV_MENU_ROOT_BACK_BTN_DISABLED`
- `LV_MENU_ROOT_BACK_BTN_ENABLED`

You can set root back button modes with `lv_menu_set_mode_root_back_btn(menu, LV_MENU_ROOT_BACK_BTN...)`.

## Create a menu page

`lv_menu_page_create(menu, title)` creates a new empty menu page. You can add any widgets to the page.

## Set a menu page in the main area

Once a menu page has been created, you can set it to the main area with `lv_menu_set_page(menu, page)`. `NULL` to clear main and clear menu history.

## Set a menu page in the sidebar

Once a menu page has been created, you can set it to the sidebar with `lv_menu_set_sidebar_page(menu, page)`. `NULL` to clear sidebar.

## Linking between menu pages

For instance, you have created a btn obj in the main page. When you click the btn obj, you want it to open up a new page, use `lv_menu_set_load_page_event(menu, obj, new page)`.

## Create a menu container, section, separator

The following objects can be created so that it is easier to style the menu:

`lv_menu_cont_create(parent page)` creates a new empty container.

`lv_menu_section_create(parent page)` creates a new empty section.

`lv_menu_separator_create(parent page)` creates a separator.

## Events

- `LV_EVENT_VALUE_CHANGED` Sent when a page is shown.
  - `lv_menu_get_cur_main_page(menu)` returns a pointer to menu page that is currently displayed in main.
  - `lv_menu_get_cur_sidebar_page(menu)` returns a pointer to menu page that is currently displayed in sidebar.
- `LV_EVENT_CLICKED` Sent when a back btn in a header from either main or sidebar is clicked. `LV_OBJ_FLAG_EVENT_BUBBLE` is enabled on the buttons so you can add events to the menu itself.
  - `lv_menu_back_btn_is_root(menu, btn)` to check if btn is root back btn

See the events of the *Base object* too.

Learn more about *Events*.

## Keys

No keys are handled by the menu widget.

Learn more about *Keys*.

## Example

## API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Meter (lv\_meter)

### Overview

The Meter widget can visualize data in very flexible ways. It can show arcs, needles, ticks lines and labels.

## Parts and Styles

- **LV\_PART\_MAIN** The background of the Meter. Uses the typical background properties.
- **LV\_PART\_TICK** The tick lines a labels using the *line* and *text* style properties.
- **LV\_PART\_INDICATOR** The needle line or image using the *line* and *img* style properties, as well as the background properties to draw a square (or circle) on the pivot of the needles. Padding makes the square larger.
- **LV\_PART\_ITEMS** The arcs using the *arc* properties.

## Usage

### Add a scale

First a *Scale* needs to be added to the Meter with `lv_meter_scale_t * scale = lv_meter_add_scale(meter)`. The Scale has minor and major ticks and labels on the major ticks. Later indicators (needles, arcs, tick modifiers) can be added to the meter

Any number of scales can be added to Meter.

The minor tick lines can be configured with: `lv_meter_set_scale_ticks(meter, scale, tick_count, line_width, tick_length, ctick_color)`.

To add major tick lines use `lv_meter_set_scale_major_ticks(meter, scale, nth_major, tick_width, tick_length, tick_color, label_gap)`. `nth_major` to specify how many minor ticks to skip to draw a major tick.

Labels are added automatically on major ticks with `label_gap` distance from the ticks with text proportionally to the values of the tick line.

`lv_meter_set_scale_range(meter, scale, min, max, angle_range, rotation)` sets the value and angle range of the scale.

### Add indicators

Indicators need to be added to a Scale and their value is interpreted in the range of the Scale.

All the indicator add functions return `lv_meter_indicator_t *`.

### Needle line

`indic = lv_meter_add_needle_line(meter, scale, line_width, line_color, r_mod)` adds a needle line to a Scale. By default, the length of the line is the same as the scale's radius but `r_mod` changes the length.

`lv_meter_set_indicator_value(meter, indic, value)` sets the value of the indicator.

## Needle image

`indic = lv_meter_add_needle_img(meter, scale, img_src, pivot_x, pivot_y)` sets an image that will be used as a needle. `img_src` should be a needle pointing to the right like this `-0-->`. `pivot_x` and `pivot_y` sets the pivot point of the rotation relative to the top left corner of the image.

`lv_meter_set_indicator_value(meter, indicator, value)` sets the value of the indicator.

## Arc

`indic = lv_meter_add_arc(meter, scale, arc_width, arc_color, r_mod)` adds an arc indicator. By default, the radius of the arc is the same as the scale's radius but `r_mod` changes the radius.

`lv_meter_set_indicator_start_value(meter, indic, value)` and `lv_meter_set_indicator_end_value(meter, indicator, value)` sets the value of the indicator.

## Scale lines (ticks)

`indic = lv_meter_add_scale_lines(meter, scale, color_start, color_end, local, width_mod)` adds an indicator that modifies the ticks lines. If `local` is `true` the ticks' color will be faded from `color_start` to `color_end` in the indicator's start and end value range. If `local` is `false` `color_start` and `color_end` will be mapped to the start and end value of the scale and only a "slice" of that color gradient will be visible in the indicator's start and end value range. `width_mod` modifies the width of the tick lines.

`lv_meter_set_indicator_start_value(meter, indicator, value)` and `lv_meter_set_indicator_end_value(meter, indicator, value)` sets the value of the indicator.

## Events

- `LV_EVENT_DRAW_PART_BEGIN` and `LV_EVENT_DRAW_PART_END` is sent for the following types:
  - `LV_METER_DRAW_PART_ARC` The arc indicator
    - \* `part`: `LV_PART_ITEMS`
    - \* `sub_part_ptr`: pointer to the indicator
    - \* `arc_dsc`
    - \* `radius`: radius of the arc
    - \* `p1` center of the arc
  - `LV_METER_DRAW_PART_NEEDLE_LINE` The needle lines
    - \* `part`: `LV_PART_ITEMS`
    - \* `p1, p2` points of the line
    - \* `line_dsc`
    - \* `sub_part_ptr`: pointer to the indicator
  - `LV_METER_DRAW_PART_NEEDLE_IMG` The needle images
    - \* `part`: `LV_PART_ITEMS`

- \* p1, p2 points of the line
- \* img\_dsc
- \* sub\_part\_ptr: pointer to the indicator
- LV\_METER\_DRAW\_PART\_TICK The tick lines and labels
  - \* part: LV\_PART\_TICKS
  - \* value: the value of the line
  - \* text: value converted to decimal or NULL on minor lines
  - \* label\_dsc: label draw descriptor or NULL on minor lines
  - \* line\_dsc:
  - \* id: the index of the line

See the events of the *Base object* too.

Learn more about *Events*.

## Keys

No keys are handled by the Meter widget.

Learn more about *Keys*.

## Example

## API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Message box (lv\_msgbox)

### Overview

The Message boxes act as pop-ups. They are built from a background container, a title, an optional close button, a text and optional buttons.

The text will be broken into multiple lines automatically and the height will be set automatically to include the text and the buttons.

The message box can be modal (blocking clicks on the rest of the screen) or not modal.

## Parts and Styles

The message box is built from other widgets, so you can check these widgets' documentation for details.

- Background: *lv\_obj*
- Close button: *lv\_btn*
- Title and text: *lv\_label*
- Buttons: *lv\_btnmatrix*

## Usage

### Create a message box

`lv_msgbox_create(parent, title, txt, btn_txts[], add_close_btn)` creates a message box.

If `parent` is `NULL` the message box will be modal. `title` and `txt` are strings for the title and the text. `btn_txts[]` is an array with the buttons' text. E.g. `const char * btn_txts[] = {"Ok", "Cancel", NULL}`. `add_close_btn` can be `true` or `false` to add/don't add a close button.

### Get the parts

The building blocks of the message box can be obtained using the following functions:

```
lv_obj_t * lv_msgbox_get_title(lv_obj_t * mbox);  
lv_obj_t * lv_msgbox_get_close_btn(lv_obj_t * mbox);  
lv_obj_t * lv_msgbox_get_text(lv_obj_t * mbox);  
lv_obj_t * lv_msgbox_get_btns(lv_obj_t * mbox);
```

### Close the message box

`lv_msgbox_close(msgbox)` closes (deletes) the message box.

## Events

- `LV_EVENT_VALUE_CHANGED` is sent by the buttons if one of them is clicked. `LV_OBJ_FLAG_EVENT_BUBBLE` is enabled on the buttons so you can add events to the message box itself. In the event handler, `lv_event_get_target(e)` will return the button matrix and `lv_event_get_current_target(e)` will return the message box. `lv_msgbox_get_active_btn(msgbox)` and `lv_msgbox_get_active_btn_text(msgbox)` can be used to get the index and text of the clicked button.

Learn more about [Events](#).

## Keys

Keys have effect on the close button and button matrix. You can add them manually to a group if required.

Learn more about [Keys](#).

## Example

## API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Span (lv\_span)

### Overview

A spangroup is the object that is used to display rich text. Different from the label object, **spangroup** can render text styled with different fonts, colors, and sizes into the spangroup object.

### Parts and Styles

- **LV\_PART\_MAIN** The spangroup has only one part.

### Usage

#### Set text and style

The spangroup object uses span to describe text and text style. so, first we need to create span descriptor using `lv_span_t * span = lv_spangroup_new_span(spangroup)`. Then use `lv_span_set_text(span, "text")` to set text. The style of the span is configured as with a normal style object by using its `style` member, eg: `lv_style_set_text_color(&span->style, lv_palette_main(LV_PALETTE_RED))`.

If spangroup object mode `!= LV_SPAN_MODE_FIXED` you must call `lv_spangroup_refr_mode()` after you have modified span style(eg:set text, changed the font size, del span).

#### Retreiving a span child

Spangroups store their children differently from normal objects, so normal functions for getting children won't work.

`lv_spangroup_get_child(spangroup, id)` will return a pointer to the child span at index `id`. In addition, `id` can be negative to index from the end of the spangroup where `-1` is the youngest child, `-2` is second youngest, etc.

e.g. `lv_span_t* span = lv_spangroup_get_child(spangroup, 0)` will return the first child of the spangroup. `lv_span_t* span = lv_spangroup_get_child(spangroup, -1)` will return the last (or most recent) child.

## Child Count

Use the function `lv_spangroup_get_child_cnt(spangroup)` to get back the number of spans the group is maintaining.

e.g. `uint32_t size = lv_spangroup_get_child_cnt(spangroup)`

## Text align

like label object, the spangroup can be set to one the following modes:

- `LV_TEXT_ALIGN_LEFT` Align text to left.
- `LV_TEXT_ALIGN_CENTER` Align text to center.
- `LV_TEXT_ALIGN_RIGHT` Align text to right.
- `LV_TEXT_ALIGN_AUTO` Align text auto.

use function `lv_spangroup_set_align(spangroup, LV_TEXT_ALIGN_CENTER)` to set text align.

## Modes

The spangroup can be set to one the following modes:

- `LV_SPAN_MODE_FIXED` fixes the object size.
- `LV_SPAN_MODE_EXPAND` Expand the object size to the text size but stay on a single line.
- `LV_SPAN_MODE_BREAK` Keep width, break the too long lines and auto expand height.

Use `lv_spangroup_set_mode(spangroup, LV_SPAN_MODE_BREAK)` to set object mode.

## Overflow

The spangroup can be set to one the following modes:

- `LV_SPAN_OVERFLOW_CLIP` truncates the text at the limit of the area.
- `LV_SPAN_OVERFLOW_ELLIPSIS` will display an ellipsis( . . . ) when text overflows the area.

Use `lv_spangroup_set_overflow(spangroup, LV_SPAN_OVERFLOW_CLIP)` to set object overflow mode.

## first line indent

Use `lv_spangroup_set_indent(spangroup, 20)` to set the indent of the first line. all modes support pixel units, in addition to `LV_SPAN_MODE_FIXED` and `LV_SPAN_MODE_BREAK` mode supports percentage units too.



## Events

No special events are sent by this widget.

Learn more about [Events](#).

## Keys

No *Keys* are processed by the object type.

Learn more about [Keys](#).

## Example

## API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Spinbox (lv\_spinbox)

### 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](#).

### Parts and Styles

The parts of the Spinbox are identical to the [Text area](#).

### Value, range and step

`lv_spinbox_set_value(spinbox, 1234)` sets a new value on the Spinbox.

`lv_spinbox_increment(spinbox)` and `lv_spinbox_decrement(spinbox)` increments/decrements the value of the Spinbox according to the currently selected digit.

`lv_spinbox_set_range(spinbox, -1000, 2500)` sets a range. If the value is changed by `lv_spinbox_set_value`, by *Keys*, `lv_spinbox_increment/decrement` this range will be respected.

`lv_spinbox_set_step(spinbox, 100)` sets which digits to change on increment/decrement. Only multiples of ten can be set, and not for example 3.

`lv_spinbox_set_pos(spinbox, 1)` sets the cursor to a specific digit to change on increment/decrement. For example position '0' sets the cursor to the least significant digit.

If an encoder is used as input device, the selected digit is shifted to the right by default whenever the encoder button is clicked. To change this behaviour to shifting to the left, the `lv_spinbox_set_digit_step_direction(spinbox, LV_DIR_LEFT)` can be used

## Format

`lv_spinbox_set_digit_format(spinbox, digit_count, separator_position)` sets the number format. `digit_count` is the number of digits excluding the decimal separator and the sign. `separator_position` is the number of digits before the decimal point. If 0, no decimal point is displayed.

## Rollover

`lv_spinbox_set_rollover(spinbox, true/false)` enables/disabled rollover mode. If either the minimum or maximum value is reached with rollover enabled, the value will change to the other limit. If rollover is disabled the value will remain at the minimum or maximum value.

## Events

- `LV_EVENT_VALUE_CHANGED` Sent when the value has changed.

See the events of the [Text area](#) too.

Learn more about [Events](#).

## Keys

- `LV_KEY_LEFT/RIGHT` With *Keypad* move the cursor left/right. With *Encoder* decrement/increment the selected digit.
- `LV_KEY_UP/DOWN` With *Keypad* and *Encoder* increment/decrement the value.
- `LV_KEY_ENTER` With *Encoder* got the net digit. Jump to the first after the last.

## Example

### API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Example

### Spinner (lv\_spinner)

#### Overview

The Spinner object is a spinning arc over a ring.

## Parts and Styles

The parts are identical to the parts of *lv\_arc*.

## Usage

### Create a spinner

To create a spinner use `lv_spinner_create(parent, spin_time, arc_length)`. `spin_time` sets the spin time in milliseconds, `arc_length` sets the length of the spinning arc in degrees.

## Events

No special events are sent to the Spinner.

See the events of the *Arc* too.

Learn more about *Events*.

## Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

## Example

## API

**警告:** doxygenfile: Unable to find project ‘lvgl’ in breathe\_projects dictionary

## Tabview (lv\_tabview)

### Overview

The Tab view object can be used to organize content in tabs. The Tab view is built from other widgets:

- Main container: *lv\_obj*)
  - Tab buttons: *lv\_btnmatrix*
  - Container for the tabs: *lv\_obj*
    - \* Content of the tabs: *lv\_obj*

The tab buttons can be positioned on the top, bottom, left and right side of the Tab view.

A new tab can be selected either by clicking on a tab button or by sliding horizontally on the content.

## Parts and Styles

There are no special parts on the Tab view but the `lv_obj` and `lv_btnmatrix` widgets are used to create the Tab view.

## Usage

### Create a Tab view

`lv_tabview_create(parent, tab_pos, tab_size);` creates a new empty Tab view. `tab_pos` can be `LV_DIR_TOP/BOTTOM/LEFT/RIGHT` to position the tab buttons to a side. `tab_size` is the height (in case of `LV_DIR_TOP/BOTTOM`) or width (in case of `LV_DIR_LEFT/RIGHT`) tab buttons.

### Add tabs

New tabs can be added with `lv_tabview_add_tab(tabview, "Tab name")`. This will return a pointer to an *lv\_obj* object where the tab's content can be created.

### Change tab

To select a new tab you can:

- Click on its tab button
- Slide horizontally
- Use `lv_tabview_set_act(tabview, id, LV_ANIM_ON/OFF)` function

### Get the parts

`lv_tabview_get_content(tabview)` returns the container for the tabs,  
`lv_tabview_get_tab_btns(tabview)` returns the Tab buttons object which is a *Button matrix*.

## Events

- `LV_EVENT_VALUE_CHANGED` Sent when a new tab is selected by sliding or clicking the tab button.  
`lv_tabview_get_tab_act(tabview)` returns the zero based index of the current tab.

Learn more about *Events*.

## Keys

Keys have effect only on the tab buttons (Button matrix). Add manually to a group if required.

Learn more about [Keys](#).

## Example

## API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Tile view (lv\_tileview)

### Overview

The Tile view is a container object whose elements (called *tiles*) can be arranged in grid form. A user can navigate between the tiles by swiping. Any direction of swiping can be disabled on the tiles individually to not allow moving from one tile to another.

If the Tile view is screen sized, the user interface resembles what you may have seen on smartwatches.

### Parts and Styles

The Tile view is built from an [lv\\_obj](#) container and [lv\\_obj](#) tiles.

The parts and styles work the same as for [lv\\_obj](#).

### Usage

#### Add a tile

`lv_tileview_add_tile(tileview, row_id, col_id, dir)` creates a new tile on the `row_id`th row and `col_id`th column. `dir` can be `LV_DIR_LEFT/RIGHT/TOP/BOTTOM/HOR/VER/ALL` or OR-ed values to enable moving to the adjacent tiles into the given direction by swiping.

The returned value is an `lv_obj_t *` on which the content of the tab can be created.

#### Change tile

The Tile view can scroll to a tile with `lv_obj_set_tile(tileview, tile_obj, LV_ANIM_ON/OFF)` or `lv_obj_set_tile_id(tileview, col_id, row_id, LV_ANIM_ON/OFF);`

## Events

- `LV_EVENT_VALUE_CHANGED` Sent when a new tile loaded by scrolling. `lv_tileview_get_tile_act(tabview)` can be used to get current tile.

## Keys

*Keys* are not handled by the Tile view.

Learn more about [\*Keys\*](#).

## Example

## API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## Window (`lv_win`)

### Overview

The Window is container-like object built from a header with title and buttons and a content area.

### Parts and Styles

The Window is built from other widgets so you can check their documentation for details:

- Background: *lv\_obj*
- Header on the background: *lv\_obj*
- Title on the header: *lv\_label*
- Buttons on the header: *lv\_btn*
- Content area on the background: *lv\_obj*

## Usage

### Create a Window

`lv_win_create(parent, header_height)` creates a Window with an empty header.

## Title and buttons

Any number of texts (but typically only one) can be added to the header with `lv_win_add_title(win, "The title")`.

Control buttons can be added to the window's header with `lv_win_add_btn(win, icon, btn_width)`. `icon` can be any image source, and `btn_width` is the width of the button.

The title and the buttons will be added in the order the functions are called. So adding a button, a text and two other buttons will result in a button on the left, a title, and 2 buttons on the right. The width of the title is set to take all the remaining space on the header. In other words: it pushes to the right all the buttons that are added after the title.

## Get the parts

`lv_win_get_header(win)` returns a pointer to the header, `lv_win_get_content(win)` returns a pointer to the content container to which the content of the window can be added.

## Events

No special events are sent by the windows, however events can be added manually to the return value of `lv_win_add_btn`.

Learn more about [Events](#).

## Keys

No *Keys* are handled by the window.

Learn more about [Keys](#).

## Example

### API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## 1.6 布局 (Layouts)

### 1.6.1 Flex

#### Overview

The Flexbox (or Flex for short) is a subset of [CSS Flexbox](#).

It can arrange items into rows or columns (tracks), handle wrapping, adjust the spacing between the items and tracks, handle *grow* to make the item(s) fill the remaining space with respect to min/max width and height.

To make an object flex container call `lv_obj_set_layout(obj, LV_LAYOUT_FLEX)`.

Note that the flex layout feature of LVGL needs to be globally enabled with `LV_USE_FLEX` in `lv_conf.h`.

## Terms

- **tracks:** the rows or columns
- **main direction:** row or column, the direction in which the items are placed
- **cross direction:** perpendicular to the main direction
- **wrap:** if there is no more space in the track a new track is started
- **grow:** if set on an item it will grow to fill the remaining space on the track. The available space will be distributed among items respective to their grow value (larger value means more space)
- **gap:** the space between the rows and columns or the items on a track

## Simple interface

With the following functions you can set a Flex layout on any parent.

### Flex flow

`lv_obj_set_flex_flow(obj, flex_flow)`

The possible values for `flex_flow` are:

- `LV_FLEX_FLOW_ROW` Place the children in a row without wrapping
- `LV_FLEX_FLOW_COLUMN` Place the children in a column without wrapping
- `LV_FLEX_FLOW_ROW_WRAP` Place the children in a row with wrapping
- `LV_FLEX_FLOW_COLUMN_WRAP` Place the children in a column with wrapping
- `LV_FLEX_FLOW_ROW_REVERSE` Place the children in a row without wrapping but in reversed order
- `LV_FLEX_FLOW_COLUMN_REVERSE` Place the children in a column without wrapping but in reversed order
- `LV_FLEX_FLOW_ROW_WRAP_REVERSE` Place the children in a row with wrapping but in reversed order
- `LV_FLEX_FLOW_COLUMN_WRAP_REVERSE` Place the children in a column with wrapping but in reversed order

### Flex align

To manage the placement of the children use `lv_obj_set_flex_align(obj, main_place, cross_place, track_cross_place)`

- **main\_place** determines how to distribute the items in their track on the main axis. E.g. flush the items to the right on `LV_FLEX_FLOW_ROW_WRAP`. (It's called **justify-content** in CSS)
- **cross\_place** determines how to distribute the items in their track on the cross axis. E.g. if the items have different height place them to the bottom of the track. (It's called **align-items** in CSS)
- **track\_cross\_place** determines how to distribute the tracks (It's called **align-content** in CSS)

The possible values are:

- `LV_FLEX_ALIGN_START` means left on a horizontally and top vertically. (default)



- `LV_FLEX_ALIGN_END` means right on a horizontally and bottom vertically
- `LV_FLEX_ALIGN_CENTER` simply center
- `LV_FLEX_ALIGN_SPACE_EVENLY` items are distributed so that the spacing between any two items (and the space to the edges) is equal. Does not apply to `track_cross_place`.
- `LV_FLEX_ALIGN_SPACE_AROUND` items are evenly distributed in the track with equal space around them. Note that visually the spaces aren't equal, since all the items have equal space on both sides. The first item will have one unit of space against the container edge, but two units of space between the next item because that next item has its own spacing that applies. Not applies to `track_cross_place`.
- `LV_FLEX_ALIGN_SPACE_BETWEEN` items are evenly distributed in the track: first item is on the start line, last item on the end line. Not applies to `track_cross_place`.

## Flex grow

Flex grow can be used to make one or more children fill the available space on the track. When more children have grow parameters, the available space will be distributed proportionally to the grow values. For example, there is 400 px remaining space and 4 objects with grow:

- A with grow = 1
- B with grow = 1
- C with grow = 2

A and B will have 100 px size, and C will have 200 px size.

Flex grow can be set on a child with `lv_obj_set_flex_grow(child, value)`. `value` needs to be `> 1` or `0` to disable grow on the child.

## Style interface

All the Flex-related values are style properties under the hood and you can use them similarly to any other style property. The following flex related style properties exist:

- `FLEX_FLOW`
- `FLEX_MAIN_PLACE`
- `FLEX_CROSS_PLACE`
- `FLEX_TRACK_PLACE`
- `FLEX_GROW`

## Internal padding

To modify the minimum space flexbox inserts between objects, the following properties can be set on the flex container style:

- `pad_row` Sets the padding between the rows.
- `pad_column` Sets the padding between the columns.

These can for example be used if you don't want any padding between your objects:  
`lv_style_set_pad_column(&row_container_style,0)`

## Other features

### RTL

If the base direction of the container is set the `LV_BASE_DIR_RTL` the meaning of `LV_FLEX_ALIGN_START` and `LV_FLEX_ALIGN_END` is swapped on `ROW` layouts. I.e. `START` will mean right.

The items on `ROW` layouts, and tracks of `COLUMN` layouts will be placed from right to left.

### New track

You can force Flex to put an item into a new line with `lv_obj_add_flag(child, LV_OBJ_FLAG_FLEX_IN_NEW_TRACK)`.

### Example

### API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## 1.6.2 Grid

### Overview

The Grid layout is a subset of [CSS Flexbox](#).

It can arrange items into a 2D “table” that has rows or columns (tracks). The item can span through multiple columns or rows. The track’s size can be set in pixel, to the largest item (`LV_GRID_CONTENT`) or in “Free unit” (`FR`) to distribute the free space proportionally.

To make an object a grid container call `lv_obj_set_layout(obj, LV_LAYOUT_GRID)`.

Note that the grid layout feature of LVGL needs to be globally enabled with `LV_USE_GRID` in `lv_conf.h`.

### Terms

- tracks: the rows or columns
- free unit (FR): if set on track’s size is set in `FR` it will grow to fill the remaining space on the parent.
- gap: the space between the rows and columns or the items on a track

## Simple interface

With the following functions you can easily set a Grid layout on any parent.

## Grid descriptors

First you need to describe the size of rows and columns. It can be done by declaring 2 arrays and the track sizes in them. The last element must be LV\_GRID\_TEMPLATE\_LAST.

For example:

```
static lv_coord_t column_dsc[] = {100, 400, LV_GRID_TEMPLATE_LAST}; /*2 columns
↪with 100 and 400 ps width*/
static lv_coord_t row_dsc[] = {100, 100, 100, LV_GRID_TEMPLATE_LAST}; /*3 100 px tall
↪rows*/
```

To set the descriptors on a parent use `lv_obj_set_grid_dsc_array(obj, col_dsc, row_dsc)`.

Besides simple settings the size in pixel you can use two special values:

- LV\_GRID\_CONTENT set the width to the largest children on this track
- LV\_GRID\_FR(X) tell what portion of the remaining space should be used by this track. Larger value means larger space.

## Grid items

By default, the children are not added to the grid. They need to be added manually to a cell.

To do this call `lv_obj_set_grid_cell(child, column_align, column_pos, column_span, row_align, row_pos, row_span)`.

`column_align` and `row_align` determine how to align the children in its cell. The possible values are:

- LV\_GRID\_ALIGN\_START means left on a horizontally and top vertically. (default)
- LV\_GRID\_ALIGN\_END means right on a horizontally and bottom vertically
- LV\_GRID\_ALIGN\_CENTER simply center

`column_pos` and `row_pos` means the zero based index of the cell into the item should be placed.

`column_span` and `row_span` means how many tracks should the item involve from the start cell. Must be > 1.

## Grid align

If there are some empty space the track can be aligned several ways:

- LV\_GRID\_ALIGN\_START means left on a horizontally and top vertically. (default)
- LV\_GRID\_ALIGN\_END means right on a horizontally and bottom vertically
- LV\_GRID\_ALIGN\_CENTER simply center
- LV\_GRID\_ALIGN\_SPACE\_EVENLY items are distributed so that the spacing between any two items (and the space to the edges) is equal. Not applies to `track_cross_place`.

- `LV_GRID_ALIGN_SPACE_AROUND` items are evenly distributed in the track with equal space around them. Note that visually the spaces aren't equal, since all the items have equal space on both sides. The first item will have one unit of space against the container edge, but two units of space between the next item because that next item has its own spacing that applies. Not applies to `track_cross_place`.
- `LV_GRID_ALIGN_SPACE_BETWEEN` items are evenly distributed in the track: first item is on the start line, last item on the end line. Not applies to `track_cross_place`.

To set the track's alignment use `lv_obj_set_grid_align(obj, column_align, row_align)`.

## Style interface

All the Grid related values are style properties under the hood and you can use them similarly to any other style properties. The following Grid related style properties exist:

- `GRID_COLUMN_DSC_ARRAY`
- `GRID_ROW_DSC_ARRAY`
- `GRID_COLUMN_ALIGN`
- `GRID_ROW_ALIGN`
- `GRID_CELL_X_ALIGN`
- `GRID_CELL_COLUMN_POS`
- `GRID_CELL_COLUMN_SPAN`
- `GRID_CELL_Y_ALIGN`
- `GRID_CELL_ROW_POS`
- `GRID_CELL_ROW_SPAN`

## Internal padding

To modify the minimum space Grid inserts between objects, the following properties can be set on the Grid container style:

- `pad_row` Sets the padding between the rows.
- `pad_column` Sets the padding between the columns.

## Other features

### RTL

If the base direction of the container is set to `LV_BASE_DIR_RTL`, the meaning of `LV_GRID_ALIGN_START` and `LV_GRID_ALIGN_END` is swapped. I.e. `START` will mean right-most.

The columns will be placed from right to left.

## Example

## API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## 1.7 第三方库 (3rd party libraries)

### 1.7.1 File System Interfaces

LVGL has a [File system](#) module to provides an abstraction layer for various file system drivers.

LVGL has build in support for

- [FATFS](#)
- [STDIO](#) (Linux and Windows using C standard function .e.g fopen, fread)
- [POSIX](#) (Linux and Windows using POSIX function .e.g open, read)
- [WIN32](#) (Windows using Win32 API function .e.g CreateFileA, ReadFile)

You still need to provide the drivers and libraries, this extensions provide only the bridge between FATFS, STDIO, POSIX, WIN32 and LVGL.

## Usage

In `lv_conf.h` set a driver letter for one or more `LV_FS_USE_...` define(s). After that you can access files using that driver letter. Setting '`\0`' will disable use of that interface.

### 1.7.2 BMP decoder

This extension allows the use of BMP images in LVGL. This implementation uses [bmp-decoder](#) library. The pixel are read on demand (not the whole image is loaded) so using BMP images requires very little RAM.

If enabled in `lv_conf.h` by `LV_USE_BMP` LVGL will register a new image decoder automatically so BMP files can be directly used as image sources. For example:

```
lv_img_set_src(my_img, "S:path/to/picture.bmp");
```

Note that, a file system driver needs to registered to open images from files. Read more about it [here](#) or just enable one in `lv_conf.h` with `LV_USE_FS...`

## Limitations

- Only BMP files are supported and BMP images as C array (`lv_img_dsc_t`) are not. It's because there is no practical differences between how the BMP files and LVGL's image format stores the image data.
- BMP files can be loaded only from file. If you want to store them in flash it's better to convert them to C array with [LVGL's image converter](#).
- The BMP files color format needs to match with `LV_COLOR_DEPTH`. Use GIMP to save the image in the required format. Both RGB888 and ARGB888 works with `LV_COLOR_DEPTH 32`
- Palette is not supported.
- Because not the whole image is read in can not be zoomed or rotated.

## Example

### API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## 1.7.3 JPG decoder

Allow the use of JPG images in LVGL. Besides that it also allows the use of a custom format, called Split JPG (SJPG), which can be decided in more optimal way on embedded systems.

### Overview

- Supports both normal JPG and the custom SJPG formats.
- Decoding normal JPG consumes RAM with the size for the whole uncompressed image (recommended only for devices with more RAM)
- SJPG is a custom format based on "normal" JPG and specially made for LVGL.
- SJPG is 'split-jpeg' which is a bundle of small jpeg fragments with an sjpg header.
- SJPG size will be almost comparable to the jpeg file or might be a slightly larger.
- File read from file and c-array are implemented.
- SJPEG frame fragment cache enables fast fetching of lines if available in cache.
- By default the sjpg image cache will be image width \* 2 \* 16 bytes (can be modified)
- Currently only 16 bit image format is supported (TODO)
- Only the required portion of the JPG and SJPG images are decoded, therefore they can't be zoomed or rotated.

## Usage

If enabled in `lv_conf.h` by `LV_USE_SJPG` LVGL will register a new image decoder automatically so JPG and SJPG files can be directly used as image sources. For example:

```
lv_img_set_src(my_img, "S:path/to/picture.jpg");
```

Note that, a file system driver needs to be registered to open images from files. Read more about it [here](#) or just enable one in `lv_conf.h` with `LV_USE_FS...`

## Converter

### Converting JPG to C array

- Use lvgl online tool <https://lvgl.io/tools/imageconverter>
- Color format = RAW, output format = C Array

### Converting JPG to SJPG

python3 and the PIL library required. (PIL can be installed with `pip3 install pillow`)

To create SJPG from JPG:

- Copy the image to convert into `lvgl/scripts`
- `cd lvgl/scripts`
- `python3 jpg_to_sjpg.py image_to_convert.jpg`. It creates both a C file and an SJPG image.

The expected result is:

```
Conversion started...

Input:
    image_to_convert.jpg
    RES = 640 x 480

Output:
    Time taken = 1.66 sec
    bin size = 77.1 KB
    walpaper.sjpg      (bin file)
    walpaper.c         (c array)

All good!
```

## Example

### API

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## 1.7.4 PNG decoder

Allow the use of PNG images in LVGL. This implementation uses [lodepng](#) library.

If enabled in `lv_conf.h` by `LV_USE_PNG` LVGL will register a new image decoder automatically so PNG files can be directly used as any other image sources.

Note that, a file system driver needs to be registered to open images from files. Read more about it [here](#) or just enable one in `lv_conf.h` with `LV_USE_FS...`

The whole PNG image is decoded so during decoding RAM equals to `image width x image height x 4` bytes are required.

As it might take significant time to decode PNG images LVGL's [images caching](#) feature can be useful.

## Example

### API

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## 1.7.5 GIF decoder

Allow to use of GIF images in LVGL. Based on <https://github.com/lecram/gifdec>

When enabled in `lv_conf.h` with `LV_USE_GIF` `lv_gif_create(parent)` can be used to create a gif widget.

`lv_gif_set_src(obj, src)` works very similarly to `lv_img_set_src`. As source it also accepts images as variables (`lv_img_dsc_t`) or files.

## Convert GIF files to C array

To convert a GIF file to byte values array use [LVGL's online converter](#). Select "Raw" color format and "C array" Output format.



## Use GIF images from file

For example:

```
lv_gif_set_src(obj, "S:path/to/example.gif");
```

Note that, a file system driver needs to be registered to open images from files. Read more about it [here](#) or just enable one in `lv_conf.h` with `LV_USE_FS_...`

## Memory requirements

To decode and display a GIF animation the following amount of RAM is required:

- LV\_COLOR\_DEPTH 8: 3 x image width x image height
- LV\_COLOR\_DEPTH 16: 4 x image width x image height
- LV\_COLOR\_DEPTH 32: 5 x image width x image height

## Example

### API

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## 1.7.6 FreeType support

Interface to [FreeType](#) to generate font bitmaps run time.

### Install FreeType

- Download Freetype from [here](#)
- make
- sudo make install

### Add FreeType to your project

- Add include path: `/usr/include/freetype2` (for GCC: `-I/usr/include/freetype2 -L/usr/local/lib`)
- Add library: `freetype` (for GCC: `-L/usr/local/lib -lfreetype`)

## Usage

Enable `LV_USE_FREETYPE` in `lv_conf.h`.

To cache the glyphs from the opened fonts set `LV_FREETYPE_CACHE_SIZE >= 0` and then use the following macros for detailed configuration:

1. `LV_FREETYPE_CACHE_SIZE`:maximum memory(bytes) used to cache font bitmap, outline, character maps, etc. 0 means use the system default value, less than 0 means disable cache.Note: that this value does not account for managed `FT_Face` and `FT_Size` objects.
2. `LV_FREETYPE_CACHE_FT_FACES`:maximum number of opened `FT_Face` objects managed by this cache instance.0 means use the system default value.Only useful when `LV_FREETYPE_CACHE_SIZE >= 0`.
3. `LV_FREETYPE_CACHE_FT_SIZES`:maximum number of opened `FT_Size` objects managed by this cache instance. 0 means use the system default value.Only useful when `LV_FREETYPE_CACHE_SIZE >= 0`.

When you are sure that all the used fonts size will not be greater than 256, you can enable `LV_FREETYPE_SBIT_CACHE`, which is much more memory efficient for small bitmaps.

You can use `lv_ft_font_init()` to create FreeType fonts. It returns `true` to indicate success, at the same time, the `font` member of `lv_ft_info_t` will be filled with a pointer to an lvgl font, and you can use it like any lvgl font.

Font style supports bold and italic, you can use the following macro to set:

1. `FT_FONT_STYLE_NORMAL`:default style.
2. `FT_FONT_STYLE_ITALIC`:Italic style
3. `FT_FONT_STYLE_BOLD`:bold style

They can be combined.eg:`FT_FONT_STYLE_BOLD | FT_FONT_STYLE_ITALIC`.

Note that, the FreeType extension doesn't use LVGL's file system. You can simply pass the path to the font as usual on your operating system or platform.

## Example

### Learn more

- [FreeType tutorial](#)
- [LVGL's font interface](#)

## API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

### 1.7.7 QR code

QR code generation with LVGL. Uses [QR-Code-generator](#) by [nayuki](#).

## Get started

- Download or clone this repository
  - [Download](#) from GitHub
  - Clone: `git clone https://github.com/lvgl/lv_lib_qrcode.git`
- Include the library: `#include "lv_lib_qrcode/lv_qrcode.h"`
- Test with the following code:

```
const char * data = "Hello world";

/*Create a 100x100 QR code*/
lv_obj_t * qr = lv_qrcode_create(lv_scr_act(), 100, lv_color_hex3(0x33f), lv_color_
→hex3(0xeef));

/*Set data*/
lv_qrcode_update(qr, data, strlen(data));
```

## Notes

- QR codes with less data are smaller but they scaled by an integer numbers number to best fit to the given size

## Example

### API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## 1.7.8 Lottie player

Allows to use Lottie animations in LVGL. Taken from this [base repository](#)

LVGL provides the interface to [Samsung/rlottie](#) library's C API. That is the actual Lottie player is not part of LVGL, it needs to be built separately.

### Build Rlottie

To build Samsung's Rlottie C++14-compatible compiler and optionally CMake 3.14 or higher is required.

To build on desktop you can follow the instructions from Rlottie's [README](#). In the most basic case it looks like this:

```
mkdir rlottie_workdir
cd rlottie_workdir
git clone https://github.com/Samsung/rlottie.git
mkdir build
cd build
cmake ../rlottie
make -j
sudo make install
```

And finally add the `-lrlottie` flag to your linker.

On embedded systems you need to take care of integrating Rlottie to the given build system.

## Usage

You can use animation from files or raw data (text). In either case first you need to enable `LV_USE_RLOTTIE` in `lv_conf.h`.

The `width` and `height` of the object be set in the `create` function and the animation will be scaled accordingly.

## Use Rlottie from file

To create a Lottie animation from file use:

```
lv_obj_t * lottie = lv_rlottie_create_from_file(parent, width, height, "path/to/
↳lottie.json");
```

Note that, Rlottie uses the standard STDIO C file API, so you can use the path “normally” and no LVGL specific driver letter is required.

## Use Rlottie from raw string data

`lv_example_rlottie_approve.c` contains an example animation in raw format. Instead storing the JSON string a hex array is stored for the following reasons:

- avoid escaping " in the JSON file
- some compilers don't support very long strings

`lvgl/scripts/filetohex.py` can be used to convert a Lottie file a hex array. E.g.:

```
./filetohex.py path/to/lottie.json > out.txt
```

To create an animation from raw data:

```
extern const uint8_t lottie_data[];
lv_obj_t* lottie = lv_rlottie_create_from_raw(parent, width, height, (const char_
↳*)lottie_data);
```

## Getting animations

Lottie is standard and popular format so you can find many animation files on the web. For example: <https://lottiefiles.com/>

You can also create your own animations with Adobe After Effects or similar software.

## Controlling animations

LVGL provides two functions to control the animation mode: `lv_rlottie_set_play_mode` and `lv_rlottie_set_current_frame`. You'll combine your intentions when calling the first method, like in these examples:

```
lv_obj_t * lottie = lv_rlottie_create_from_file(scr, 128, 128, "test.json");
lv_obj_center(lottie);
// Pause to a specific frame
lv_rlottie_set_current_frame(lottie, 50);
lv_rlottie_set_play_mode(lottie, LV_RLOTTIE_CTRL_PAUSE); // The specified frame will
↳ be displayed and then the animation will pause

// Play backward and loop
lv_rlottie_set_play_mode(lottie, LV_RLOTTIE_CTRL_PLAY | LV_RLOTTIE_CTRL_BACKWARD | LV_
↳ RLOTTIE_CTRL_LOOP);

// Play forward once (no looping)
lv_rlottie_set_play_mode(lottie, LV_RLOTTIE_CTRL_PLAY | LV_RLOTTIE_CTRL_FORWARD);
```

The default animation mode is **play forward with loop**.

If you don't enable looping, a `LV_EVENT_READY` is sent when the animation can not make more progress without looping.

To get the number of frames in an animation or the current frame index, you can cast the `lv_obj_t` instance to a `lv_rlottie_t` instance and inspect the `current_frame` and `total_frames` members.

## Example

### API

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## 1.7.9 FFmpeg support

**FFmpeg** A complete, cross-platform solution to record, convert and stream audio and video.

### Install FFmpeg

- Download FFmpeg from [here](#)
- `./configure --disable-all --disable-autodetect --disable-podpages --disable-asm --enable-avcodec --enable-avformat --enable-decoders --enable-encoders --enable-demuxers --enable-parsers --enable-protocol='file' --enable-swscale --enable-zlib`
- `make`
- `sudo make install`

## Add FFmpeg to your project

- Add library: FFmpeg (for GCC: `-lavformat -lavcodec -lavutil -lswscale -lm -lz -lpthread`)

## Usage

Enable `LV_USE_FFMPEG` in `lv_conf.h`.

See the examples below.

Note that, the FFmpeg extension doesn't use LVGL's file system. You can simply pass the path to the image or video as usual on your operating system or platform.

## Example

### API

**警告:** doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## 1.8 其他 (Others)

### 1.8.1 Snapshot

Snapshot provides APIs to take snapshot image for LVGL object together with its children. The image will look exactly like the object.

## Usage

Simply call API `lv_snapshot_take` to generate the image descriptor which can be set as image object src using `lv_img_set_src`.

Note, only below color formats are supported for now:

- `LV_IMG_CF_TRUE_COLOR_ALPHA`
- `LV_IMG_CF_ALPHA_1BIT`
- `LV_IMG_CF_ALPHA_2BIT`
- `LV_IMG_CF_ALPHA_4BIT`
- `LV_IMG_CF_ALPHA_8BIT`

## Free the Image

The memory `lv_snapshot_take` uses are dynamically allocated using `lv_mem_alloc`. Use API `lv_snapshot_free` to free the memory it takes. This will firstly free memory the image data takes, then the image descriptor.

Take caution to free the snapshot but not delete the image object. Before free the memory, be sure to firstly unlink it from image object, using `lv_img_set_src(NULL)` and `lv_img_cache_invalidate_src(src)`.

Below code snippet explains usage of this API.

```
void update_snapshot(lv_obj_t * obj, lv_obj_t * img_snapshot)
{
    lv_img_dsc_t* snapshot = (void*)lv_img_get_src(img_snapshot);
    if(snapshot) {
        lv_snapshot_free(snapshot);
    }
    snapshot = lv_snapshot_take(obj, LV_IMG_CF_TRUE_COLOR_ALPHA);
    lv_img_set_src(img_snapshot, snapshot);
}
```

## Use Existing Buffer

If the snapshot needs update now and then, or simply caller provides memory, use API `lv_res_t lv_snapshot_take_to_buf(lv_obj_t * obj, lv_img_cf_t cf, lv_img_dsc_t * dsc, void * buf, uint32_t buff_size);` for this case. It's caller's responsibility to alloc/free the memory.

If snapshot is generated successfully, the image descriptor is updated and image data will be stored to provided `buf`.

Note that snapshot may fail if provided buffer is not enough, which may happen when object size changes. It's recommended to use API `lv_snapshot_buf_size_needed` to check the needed buffer size in byte firstly and resize the buffer accordingly.

## Example

### API

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## 1.8.2 Monkey

A simple monkey test. Use random input to stress test the application.

## Usage

Enable `LV_USE_MONKEY` in `lv_conf.h`.

First configure monkey, use `lv_monkey_config_t` to define the configuration structure, set the `type` (check *input devices* for the supported types), and then set the range of `period_range` and `input_range`, the monkey will output random operations at random times within this range. Call `lv_monkey_create` to create monkey. Finally call `lv_monkey_set_enable(monkey, true)` to enable monkey.

If you want to pause the monkey, call `lv_monkey_set_enable(monkey, false)`. To delete the monkey, call `lv_monkey_del(monkey)`.

Note that `input_range` has different meanings in different `type`:

- `LV_INDEV_TYPE_POINTER` No effect, click randomly within the pixels of the screen resolution.
- `LV_INDEV_TYPE_ENCODER` The minimum and maximum values of `enc_diff`.
- `LV_INDEV_TYPE_BUTTON` The minimum and maximum values of `btn_id`. Use `lv_monkey_get_indev()` to get the input device, and use `lv_indev_set_button_points()` to map the key ID to the coordinates.
- `LV_INDEV_TYPE_KEYPAD` No effect, Send random *Keys*.

## Example

### API

警告: doxygenfile: Unable to find project 'lvgl' in breathe\_projects dictionary

## 1.9 贡献 (Contributing)

### 1.9.1 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 understanding your contribution or issue easier and you will get a useful response 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.

### 1.9.2 Pull request

Merging new code into the 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 what 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).

#### 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.

#### 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](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 your changes to the forked lvgl repository.
5. Create a PR on GitHub from the page of your lvgl repository ([https://github.com/<YOUR\\_NAME>/lvgl](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.

## Commit message format

The commit messages format is inspired by [Angular Commit Format](#).

The following structure should be used:

```
<type>(<scope>): <subject>
<BLANK LINE>
<body>
<BLANK LINE>
<footer>
```

Possible **<type>**s:

- **fix** bugfix in the source code.
- **feat** new feature
- **arch** architectural changes
- **perf** changes that affect the performance
- **example** anything related to examples (even fixes and new examples)
- **docs** anything related to the documentation (even fixes, formatting, and new pages)
- **test** anything related to tests (new and updated tests or CI actions)
- **chore** any minor formatting or style changes that would make the changelog noisy

**<scope>** is the module, file, or sub-system that is affected by the commit. It's usually one word and can be chosen freely. For example **img**, **layout**, **txt**, **anim**. The scope can be omitted.

**<subject>** contains a short description of the change:

- use the imperative, present tense: “change” not “changed” nor “changes”
- don't capitalize the first letter
- no dot (.) at the end
- max 90 characters

**<body>** optional and can be used to describe the details of this change.

**<footer>** shall contain

- the words “BREAKING CHANGE” if the changes break the API
- reference to the GitHub issue or Pull Request if applicable.

Some examples:

```
fix(img): update size if a new source is set
```

```
fix(bar): fix memory leak
```

```
The animations weren't deleted in the destructor.
```

```
Fixes: #1234
```

```
feat: add span widget
```

```
The span widget allows mixing different font sizes, colors and styles.
It's similar to HTML <span>
```

```
docs(porting): fix typo
```

### 1.9.3 Developer Certification of Origin (DCO)

#### Overview

To ensure all licensing criteria are met for every repository 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 agree that your contribution complies with the DCO.

If your contribution fulfills the requirements of the DCO no further action is needed. If you are unsure feel free to ask us in a comment.

#### 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 MIT licensed, other MIT licensed code can be integrated without issues. The MIT license requires a copyright notice be added to the derived work. Any derivative work based on MIT licensed code must copy the original work's license file or text.

## Use GPL licensed code

The GPL license is not compatible with the MIT license. Therefore, LVGL can not accept GPL licensed code.

## 1.9.4 Ways to contribute

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 LVGL code.

Below we have collected some opportunities about the ways you can contribute to LVGL.

### 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!

### Tell what you have achieved

Have you already started using LVGL in a *Simulator*, a development board, or on your custom hardware? Was it easy or were there some obstacles? Are you happy with the result? Showing your project to others is a win-win situation because it increases your and LVGL's reputation at the same time.

You can post about your project on Twitter, Facebook, LinkedIn, create a YouTube video, and so on. Only one thing: On social media don't forget to add a link to <https://lvgl.io> or <https://github.com/lvgl> and use the hashtag #lvgl. Thank you! :)

You can also open a new topic in the [My projects](#) category of the Forum.

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.

Any of these help to spread the word and familiarize new developers with LVGL.

If you don't want to speak about your project publicly, feel free to use [Contact form](#) on [lvgl.io](https://lvgl.io) to private message to us.

### Write examples

As you learn LVGL you will probably play with the features of widgets. Why not publish your experiments?

Each widgets' documentation contains examples. For instance, here are the examples of the [Drop-down list](#) widget. The examples are directly loaded from the [lvgl/examples](#) folder.

So all you need to do is send a [Pull request](#) to the [lvgl](#) repository and follow some conventions:

- Name the examples like `lv_example_<widget_name>_<index>`.
- Make the example as short and simple as possible.
- Add comments to explain what the example does.

- Use 320x240 resolution.
- Update `index.rst` in the example's folder with your new example. To see how other examples are added, look in the `lvgl/examples/widgets` folder.

## Improve the docs

As you read the documentation you might see some typos or unclear sentences. All the documentation is located in the `lvgl/docs` folder. For typos and straightforward fixes, you can simply edit the file on GitHub.

Note that the documentation is also formatted in [Markdown](#).

## 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 out the template. It helps find the root of the problem while avoiding extensive questions and exchanges with other developers.

## 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`.

## 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 had 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!

## Add features

If you have created a cool widget, or added useful feature to LVGL feel free to open a new PR for it. We collect the optional features (a.k.a. plugins) in `lvgl/src/extra` folder so if you are interested in adding a new features please use this folder. The [README](#) file describes the basics rules of contribution and also lists some ideas.

For further ideas take a look at the [Roadmap](#) page. If you are interested in any of them feel free to share your opinion and/or participate in the implementation.

Other features which are (still) not on the road map are listed in the [Feature request](#) category of the Forum.

When adding a new features the followings also needs to be updated:

- Update `lv_conf_template.h`
- Add description in the `docs`
- Add `examples`
- Update the `changelog`

## 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 input has higher impact when we are making 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.

## 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 fit well into the LVGL organization, we will open a repository for your project where you will have admin rights.

To make this concept sustainable there 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 1 year it will be archived

## 1.10 更新日志 (Changelog)

### 1.10.1 v8.1.0 10 November 2021

#### Overview

v8.1 is minor release so besides many fixes it contains a lot of new features too.

Some of the most important features are

- Built in support for SDL based GPU drawing
- Much faster circle drawing in the software renderer
- Several 3rd party libraries are merged directly into LVGL.
- Add LVGL as an RT-Thread and ESP32 component

#### Breaking Changes

- :warning: feat(calendar): add the header directly into the calendar widget [2e08f80](#)

#### Architectural

- arch add small 3rd party libs to lvgl [2569](#)

#### New Features

- feat(display) add direct\_mode drawing mode [2460](#)
- feat(conf): make LV\_MEM\_BUF\_MAX\_NUM configurable [2747](#)
- feat(disp): add non-fullscreen display utilities [2724](#)
- feat(rlottie) add LVGL-Rlottie interface as 3rd party lib [2700](#)
- feat(rtthread): prepare for porting the device-driver of rt-thread [2719](#)
- feat(fsdv) add driver based on Win32 API [2701](#)
- feat(span) indent supports percent for fix and break mode [2693](#)
- feat(rt-thread): implement rt-thread sconscript [2674](#)
- feat(lv\_spinbox) support both right-to-left and left-to-right digit steps when clicking encoder button [2644](#)
- feat add support for rt-thread RTOS [2660](#)
- feat(disp): Enable rendering to display subsection [2583](#)
- feat(keyboard): add user-defined modes [2651](#)
- feat(event) add LV\_EVENT\_CHILD\_CREATED/DELETED [2618](#)
- feat(btnmatrix/keyboard): add option to show popovers on button press [2537](#)
- feat(msgbox) add a content area for custom content [2561](#)
- feat(tests): Include debug information to test builds [2568](#)
- feat(drawing) hardware accelerated rendering by SDL2 [2484](#)

- feat(msgbox): omit title label unless needed [2539](#)
- feat(msgbox): add function to get selected button index [2538](#)
- feat(make) add lvgl interface target for micropython [2529](#)
- feat(obj) add lv\_obj\_move\_to\_index(obj, index), renamed lv\_obj\_get\_child\_id(obj) to lv\_obj\_get\_index(obj) [2514](#)
- feat(obj) add lv\_obj\_swap() function [2461](#)
- feat(mem) LV\_MEM\_POOL\_ALLOC [2458](#)
- feat(switch) add smooth animation when changing state [2442](#)
- feat(anim) add interface for handling lv\_anim user data. [2415](#)
- feat(obj) add lv\_is\_initialized [2402](#)
- feat(obj) Backport keypad and encoder scrolling from v7 lv\_page to v8 lv\_obj [2390](#)
- feat(snapshot) add API to take snapshot for object [2353](#)
- feat(anim) add anim timeline [2309](#)
- feat(span) Add missing spangroup functions [2379](#)
- feat(img) add img\_size property [2284](#)
- feat(calendar) improve MicroPython example [2366](#)
- feat(spinbox ) add function to set cursor to specific position [2314](#)
- feat(timer) check if lv\_tick\_inc is called [aa6641a](#)
- feat(event, widgets) improve the paramter of LV\_EVENT\_DRAW\_PART\_BEGIN/END [88c4859](#)
- feat(docs) improvements to examples [4b8c73a](#)
- feat(obj) send LV\_EVENT\_DRAW\_PART\_BEGIN/END for MAIN and SCROLLBAR parts [b203167](#)
- feat(led) send LV\_EVENT\_DRAW\_PART\_BEGIN/END [fcd4aa3](#)
- feat(chart) send LV\_EVENT\_DRAW\_PART\_BEGIN/END before/after the division line drawing section. [e0ae2aa](#)
- feat(tests) upload coverage to codecov [4fff99d](#)
- feat(conf) add better check for Kconfig default [f8fe536](#)
- feat(draw) add LV\_BLEND\_MODE\_MULTIPLY [cc78ef4](#)
- feat(test) add assert for screenshot compare [2f7a005](#)
- feat(event) pass the scroll aniamtion to LV\_EVENT\_SCROLL\_BEGIN [ca54ecf](#)
- feat(obj) place the scrollbar to the left with RTL base dir. [906448e](#)
- feat(log) allow overwriting LV\_LOG\_...macros [17b8a76](#)
- feat(arc) add support to LV\_OBJ\_FLAG\_ADV\_HITTEST [dfa4f5c](#)
- feat(event) add LV\_SCREEN\_(UN)LOAD\_START [7bae9e3](#)
- feat(obj) add lv\_obj\_del\_delayed() [c6a2e15](#)
- feat(docs) add view on GitHub link [a716ac6](#)
- feat(event) add LV\_EVENT\_SCREEN\_LOADED/UNLOADED events [ee5369e](#)
- feat(textarea) remove the need of lv\_textarea\_set\_align [56ebb1a](#)



- feat(rt-thread): support LVGL projects with GCC/Keil(AC5)/Keil(AC6)/IAR [32d33fe](#)
- feat(docs) lazy load individual examples as well [918d948](#)
- feat: add LV\_USE\_MEM\_PERF/MONITOR\_POS [acd0f4f](#)
- feat(canvas) add lv\_canvas\_set\_px\_opa [b3b3ffc](#)
- feat(event) add lv\_obj\_remove\_event\_cb\_with\_user\_data [4eddeb3](#)
- feat(obj) add lv\_obj\_get\_x/y\_aligned [98bc1fe](#)

## Performance

- perf(draw) reimplement circle drawing algorithms [2374](#)
- perf(anim\_timeline) add lv\_anim\_timeline\_stop() [2411](#)
- perf(obj) remove lv\_obj\_get\_child\_cnt from cycle limit checks [ebb9ce9](#)
- perf(draw) reimplement rectangle drawing algorithms [5b3d3dc](#)
- perf(draw) ignore masks if they don't affect the current draw area [a842791](#)
- perf(refresh) optimize where to wait for lv\_disp\_flush\_ready with 2 buffers [d0172f1](#)
- perf(draw) speed up additive blending [3abe517](#)

## Fixes

- fix(bidi): add weak characters to the previous strong character's run [2777](#)
- fix(draw\_img): radius mask doesn't work in specific condition [2786](#)
- fix(border\_post): ignore bg\_img\_opa draw when draw border\_post [2788](#)
- fix(refresh) switch to portable format specifiers [2781](#)
- fix(stm32) Mark unused variable in stm32 DMA2D driver [2782](#)
- fix(conf): Make LV\_COLOR\_MIX\_ROUND\_OFS configurable [2766](#)
- fix(misc): correct the comment and code style [2769](#)
- fix(draw\_map) use existing variables instead function calls [2776](#)
- fix(draw\_img): fix typos in API comments [2773](#)
- fix(draw\_img):radius Mask doesn't work in Specific condition [2775](#)
- fix(proto) Remove redundant prototype declarations [2771](#)
- fix(conf) better support bool option from Kconfig [2555](#)
- fix(draw\_border):draw error if radius == 0 and parent clip\_corner == true [2764](#)
- fix(msgbox) add declaration for lv\_msgbox\_content\_class [2761](#)
- fix(core) add L suffix to enums to ensure 16-bit compatibility [2760](#)
- fix(anim): add lv\_anim\_get\_playtime [2745](#)
- fix(area) minor fixes [2749](#)
- fix(mem): ALIGN\_MASK should equal 0x3 on 32bit platform [2748](#)
- fix(template) prototype error [2755](#)

- fix(anim): remove time\_orig from lv\_anim\_t 2744
- fix(draw\_rect):bottom border lost if enable clip\_corner 2742
- fix(anim) and improvement 2738
- fix(draw border):border draw error if border width > radius 2739
- fix(fsdrv): remove the seek call in fs\_open 2736
- fix(fsdrv): skip the path format if LV\_FS\_XXX\_PATH not defined 2726
- fix: mark unused variable with LV\_UNUSED(XXX) instead of (void)xxx 2734
- fix(fsdrv): fix typo error in commit 752fba34f677ad73aee 2732
- fix(fsdrv): return error in case of the read/write failure 2729
- fix(refr) silence compiler warning due to integer type mismatch 2722
- fix(fs): fix the off-by-one error in the path function 2725
- fix(timer): remove the code duplication in lv\_timer\_exec 2708
- fix(async): remove the wrong comment from lv\_async\_call 2707
- fix(kconfig): change CONFIG\_LV\_THEME\_DEFAULT\_FONT to CONFIG\_LV\_FONT\_DEFAULT 2703
- fix add MP support for LVGL 3rd party libraries 2666
- fix(png) memory leak for sjpg and use lv\_mem\_... in lv\_png 2704
- fix(gif) unified whence and remove off\_t 2690
- fix(rt-thread): include the rt-thread configuration header file 2692
- fix(rt-thread): fix the ci error 2691
- fix(fsdrv) minor fs issue 2682
- fix(hal) fix typos and wording in docs for lv\_hal\_indev.h 2685
- fix(hal tick): add precompile !LV\_TICK\_CUSTOM for global variables and lv\_tick\_inc() 2675
- fix(anim\_timeline) avoid calling lv\_anim\_del(NULL, NULL) 2628
- fix(kconfig) sync Kconfig with the latest lv\_conf\_template.h 2662
- fix(log) reduce the stack usage in log function 2649
- fix(conf) make a better style alignment in lv\_conf\_internal.h 2652
- fix(span) eliminate warning in lv\_get\_snippet\_cnt() 2659
- fix(config): remove the nonexistent Kconfig 2654
- fix(Kconfig): add LV\_MEM\_ADDR config 2653
- fix(log): replace printf with fwrite to save the stack size 2655
- fix typos 2634
- fix LV\_FORMAT\_ATTRIBUTE fix for gnu > 4.4 2631
- fix(meter) make lv\_meter\_indicator\_type\_t of type uint8\_t 2632
- fix(span):crash if span->txt = "" 2616
- fix(dis) set default theme also for non-default displays 2596
- fix(label):LONG\_DOT mode crash if text Utf-8 encode > 1 2591

- fix( example) in lv\_example\_scroll\_3.py float\_btn should only be created once 2602
- fix lv\_deinit when LV\_USE\_GPU\_SDL is enabled 2598
- fix add missing LV\_ASSERT\_OBJ checks 2575
- fix(lv\_conf\_internal\_gen.py) formatting fixes on the generated file 2542
- fix(span) opa bug 2584
- fix(snapshot) snapshot is affected by parent's style because of wrong coords 2579
- fix(label):make draw area contain ext\_draw\_size 2587
- fix(btnmatrix): make ORed values work correctly with lv\_btnmatrix\_has\_btn\_ctrl 2571
- fix compiling of examples when cmake is used 2572
- fix(lv\_textarea) fix crash while delete non-ascii character in pwd mode 2549
- fix(lv\_log.h): remove the duplicated semicolon from LV\_LOG\_xxx 2544
- fix(zoom) multiplication overflow on 16-bit platforms 2536
- fix(sprintf) use \_\_has\_include for more accurate limits information 2532
- fix(font) add assert in lv\_font.c if the font is NULL 2533
- fix(lv\_types.h): remove c/c++ compiler version check 2525
- fix(lv\_utils.c): remove the unneeded header inclusion 2526
- fix(Kconfig) fix the comment in LV\_THEME\_DEFAULT\_DARK 2524
- fix(sprintf) add format string for rp2 port 2512
- fix(span) fix some bugs (overflow,decor,align) 2518
- fix(color) Bad cast in lv\_color\_mix() caused UB with 16bpp or less 2509
- fix(imgbtn) displayed incorrect when the coordinate is negative 2501
- fix(event) be sure to move all elements in copy "lv\_obj\_remove\_event\_cb" 2492
- fix(draw) use correct pointer in lv\_draw\_mask assertion 2483
- feat(mem) LV\_MEM\_POOL\_ALLOC 2458
- fix(cmake) require 'main' for Micropython 2444
- fix(docs) add static keyword to driver declaration 2452
- fix(build) remove main component dependency 2420
- fix circle drawing algorithms 2413
- fix(docs) wrong spelling of words in pictures 2409
- fix(chart) fixed point-following cursor during vertical scroll in charts 2400
- fix(chart) fixed cursor positioning with large Y rescaling without LV\_USE\_LARGE\_COORD 2399
- fix(grid.h) typos 2395
- fix(anim\_timeline) heap use after free 2394
- fix(snapshot) add missing import on MicroPython example 2389
- fix(dispatch) Fix assert failure in lv\_disp\_remove 2382
- fix(span) modify the underline position 2376

- fix(color) remove extraneous \_LV\_COLOR\_MAKE\_TYPE\_HELPER 2372
- fix(spinner) should not be clickable 2373
- fix(workflow) silence SDL warning for MicroPython 2367
- fix (span) fill LV\_EVENT\_GET\_SELF\_SIZE 2360
- fix(workflow) change MicroPython workflow to use master 2358
- fix(disp) fix memory leak in lv\_disp\_remove 2355
- fix(lv\_obj.h) typos 2350
- fix(obj) delete useless type conversion 2343
- fix(lv\_obj\_scroll.h) typos 2345
- fix(txt) enhance the function of break\_chars 2327
- fix(vglite): update for v8 e3e3eea
- fix(widgets) use lv\_obj\_class for all the widgets 3fb8baf
- fix(refr) reduce the nesting level in lv\_refr\_area 2df1282
- fix(pxp): update for v8 8a2a4a1
- fix(obj) move clean ups from lv\_obj\_del to lv\_obj\_destructor b063937
- fix (draw) fix arc bg image drawing with full arcs c3b6c6d
- fix(pxp): update RTOS macro for SDK 2.10 00c3eb1
- fix(textarea) style update in oneline mode + improve scroll to cursor 60d9a5e
- feat(led) send LV\_EVENT\_DRAW\_PART\_BEGIN/END fcd4aa3
- fix warnigs introduced by 3fb8baf5 e302403
- fix(roller) fix partial redraw of the selected area 6bc40f8
- fix(flex) fix layout update and invalidation issues 5bd82b0
- fix(indev) focus on objects on release instead of press 76a8293
- fix tests 449952e
- fix(dropdown) forget the selected option on encoder longpress e66b935
- fix(obj) improve how the focusing indev is determined a04f2de
- fix(workflow) speed up MicroPython workflow 38ad5d5
- fix(test) do not including anything in test files when not running tests 9043860
- fix tests 36b9db3
- fix(scroll) fire LV\_EVENT\_SCROLL\_BEGIN in the same spot for both axes b158932
- fix(btnmatrix) fix button invalidation on focus change 77cedfa
- fix(tlsf) do not use <assert.h> c9745b9
- fix(template) include lvgl.h in lv\_port\_\*\_template.c files 0ae15bd
- fix(docs) add margin for example description b5f632e
- fix(imgbtn) use the correct src in LV\_EVENT\_GET\_SELF\_SIZE 04c515a
- fix(color) remove extraneous cast for 8-bit color 157534c

- fix(workflow) use same Unix port variant for MicroPython submodules [ac68b10](#)
- fix(README) improve grammar [de81889](#)
- fix(printf) skip defining attribute if pycparser is used [ee9bbea](#)
- fix(README) spelling correction [41869f2](#)
- fix(color) overflow with 16 bit color depth [fe6d8d7](#)
- fix(docs) consider an example to be visible over a wider area [145a0fa](#)
- fix(codecov) disable uploading coverage for pull requests [27d88de](#)
- fix(arc) disable LV\_OBJ\_FLAG\_SCROLL\_CHAIN by default [f172eb3](#)
- fix(template) update lv\_objx\_template to v8 [38bb8af](#)
- fix(align) avoid circular references with LV\_SIZE\_CONTENT [038b781](#)
- fix(draw) with additive blending with 32 bit color depth [786db2a](#)
- fix(arc) fix arc invalidation again [5ced080](#)
- fix(align) fix lv\_obj\_align\_to [93b38e9](#)
- fix(scroll) keep the scroll position on object deleted [52edbb4](#)
- fix(dropdown) handle LV\_KEY\_ENTER [8a50edd](#)
- fix various minor warnings [924bc75](#)
- fix(textarea) various cursor drawing fixes [273a0eb](#)
- fix(label) consider base dir lv\_label\_get\_letter\_pos in special cases [6df5122](#)
- fix(imgbtn) add lv\_imgbtn\_set\_state [26e15fa](#)
- fix(printf) add (int) casts to log messages to avoid warnings on %d [d9d3f27](#)
- fix(test) silence make [7610d38](#)
- fix(test) silence make [37fd9d8](#)
- fix(calendar) update the MP example [0bab4a7](#)
- fix(scroll) fix scroll\_area\_into\_view with objects larger than the parent [5240fdd](#)
- fix(msgbox) handle NULL btn map paramter [769c4a3](#)
- fix (scroll) do not send unnecessary scroll end events [3ce5226](#)
- fix(obj\_pos) consider all alignments in content size calculation but only if x and y = 0 [5b27ebb](#)
- fix(img decoder) add error handling if the dsc->data = NULL [d0c1c67](#)
- fix(txt): skip basic arabic vowel characters when processing conjunction [5b54800](#)
- fix(typo) rename LV\_OBJ\_FLAG\_SNAPABLE to LV\_OBJ\_FLAG\_SNAPPABLE [e697807](#)
- fix(lv\_printf.h): to eliminate the errors in Keil and IAR [f6d7dc7](#)
- fix(draw) fix horizontal gradient drawing [4c034e5](#)
- fix(dropdown) use LV\_EVENT\_READY/CANCEL on list open/close [4dd1d56](#)
- fix(table) clip overflowing content [8c15933](#)
- fix(test) add #if guard to exclude test related files from the build [c12a22e](#)
- fix(test) add #if guard to exclude test related files from the build [fc364a4](#)

- fix(freetype) fix underline calculation [76c8ee6](#)
- fix(style) refresh ext. draw pad for padding and bg img [37a5d0c](#)
- fix(draw) underflow in subpixel font drawing [6d5ac70](#)
- fix(scrollbar) hide the scrollbar if the scrollble flag is removed [188a946](#)
- fix(color): minor fixes(#2767) [a4978d0](#)
- fix(group) skip object if an of the parents is hidden [5799c10](#)
- fix(obj) fix size invalidation issue on padding change [33ba722](#)
- fix(label) do not bidi process text in lv\_label\_ins\_text [e95efc1](#)
- fix(refr) set disp\_drv->draw\_buf->flushing\_last correctly with sw rotation [c514bdd](#)
- fix(draw) fix drawing small arcs [8081599](#)
- fix(chart) invalidation with LV\_CHART\_UPDATE\_MODE\_SHIFT [d61617c](#)
- fix(build) fix micropython build error [54338f6](#)
- fix(draw) fix border width of simple (radius=0, no masking) borders [20f1867](#)
- fix(calendar) fix caluculation today and highlighted day [8f0b5ab](#)
- fix(style) initialize colors to black instead of zero [524f8dd](#)
- fix(sjpg) remove unnecessary typedefs [c2d93f7](#)
- fix(label) fix clipped italic letters [2efa6dc](#)
- fix(draw) shadow darwing with large shadow width [f810265](#)
- fix(fropdown) add missing invalifations [33b5d4a](#)
- fix(dropdown) adjust the handling of keys sent to the dropdown [e41c507](#)
- fix(dis) be sure the pending scr load animation is finished in lv\_scr\_load\_anim [eb6ae52](#)
- fix(color) fox color premult precision with 16 bit color depth [f334226](#)
- fix(obj\_pos) save x,y even if the object is on a layout [a9b660c](#)
- fix(scrollbar) hide the scrollbar if the scrollble flag is removed [d9c6ad0](#)
- fix(dropdown) fix list position with RTL base direction [79edb37](#)
- fix(obj) fix lv\_obj\_align\_to with RTL base direction [531afcc](#)
- fix(chart) fix sending LV\_EVENT\_DRAW\_PART\_BEGIN/END for the cursor [34b8cd9](#)
- fix(arduino) fix the prototype of my\_touchpad\_read in the LVGL\_Arduino.ino [1a62f7a](#)
- fix(checkbox) consider the bg border when positioning the indicator [a39dac9](#)
- fix(dropdown) send LV\_EVENT\_VALUE\_CHANGED to allow styling of the list [dae7039](#)
- fix(group) fix infinite loop [bdce0bc](#)
- fix(keyboard) use LVGL heap functions instead of POSIX [b20a706](#)
- fix(blend) fix green channel with additive blending [78158f0](#)
- fix(btnmatrix) do not show pressed, focused or focus key states on disabled buttons [3df2a74](#)
- fix(font) handle the last pixel of the glyphs in font loader correctly [fa98989](#)
- fix(table) fix an off-by-one issue in self size calculation [ea2545a](#)

- fix shadowed variable [e209260](#)
- fix shadowed variable [df60018](#)
- fix(chart) be sure the chart doesn't remain scrolled out on zoom out [ad5b1bd](#)
- fix(docs) commit to meta repo as lvgl-bot instead of actual commit author [f0e8549](#)
- fix(table) invalidate the table on cell value change [cb3692e](#)
- fix(group) allow refocusing objects [1520208](#)
- fix(tabview) fix with left and right tabs [17c5744](#)
- fix(msgbox) create modals on top layer instead of act screen [5cf6303](#)
- fix(theme) show disabled state on buttons of btnmatrix, msgbox and keyboard [0be582b](#)
- fix(label) update lv\_label\_get\_letter\_pos to work with LV\_BASE\_DIR\_AUTO too [580e05a](#)
- fix(label) fix in lv\_label\_get\_letter\_pos with when pos==line\_start [58f3f56](#)
- fix(gif) replace printf statement with LVGL logging [56f62b8](#)
- fix(docs) add fsdrv back [64527a5](#)
- fix(table) remove unnecessary invalidation on pressing [6f90f9c](#)
- fix(chart) draw line chart indicator (bullet) [fba37a3](#)
- fix(anim) return the first anim if exec\_cb is NULL in lv\_anim\_get() [fb7ea10](#)
- fix(label) fix lv\_label\_get\_letter\_on with BIDI enabled [192419e](#)
- fix(checkbox) add missing invalifations [bb39e9d](#)
- fix(draw) fix gradient calculation of the rectangle is clipped [13e3470](#)
- fix(chart) fix typo in 655f42b8 [6118d63](#)
- fix(eaxmple) fix lv\_example\_chart\_2 [89081c2](#)
- fix(calendar) fix the position calculation today [ad05e19](#)
- fix(tick) minor optmization on lv\_tick\_inc call test [b4305df](#)
- fix(docs) use let instead of const for variable which gets changed [3cf5751](#)
- fix(theme) fix the switch style in the default theme [0c0dc8e](#)
- fix(tlsf) undef printf before define-ing it [cc935b8](#)
- fix(msgbox) prevent the buttons being wider than the msbgox [73e036b](#)
- fix(chart) don't draw series lines with < 1 points [655f42b](#)
- fix(tests) remove src/test\_runners when cleaning [6726b0f](#)
- fix(label) remove dupliacted lv\_obj\_refresh\_self\_size [a070ecf](#)
- fix(colowheel) disable LV\_OBJ\_FLAG\_SCROLL\_CHAIN by default [48d1c29](#)
- fix(obj) do not set the child's position in lv\_obj\_set\_parent [d89a5fb](#)
- feat: add LV\_USE\_MEM\_PERF/MONITOR\_POS [acd0f4f](#)
- fix(scroll) in scroll to view functions respect disabled LV\_OBJ\_FLAG\_SCROLLABLE [9318e02](#)
- fix(flex) remove unused variable [747b6a2](#)
- feat(canvas) add lv\_canvas\_set\_px\_opa [b3b3ffc](#)

- fix(textarea) allow using cursor with not full bg\_opa [c9d3965](#)
- fix(txt) \_lv\_txt\_get\_next\_line return 0 on empty texts [82f3fbc](#)
- fix(btnmatrix) always update row\_cnt [86012ae](#)
- fix(scroll) minor fixes on obj scroll handling [a4128a8](#)
- fix(table) consider border width for cell positions [f2987b6](#)
- fix(log) be sure LV\_LOG\_... is not empty if logs are disabled [47734c4](#)
- fix(arc) fix LV\_ARC\_MODE\_REVERSE [df3b969](#)
- fix(obj) in lv\_obj\_move\_to\_index() do not send LV\_EVENT\_CHILD\_CHANGED on all changed child [32e8276](#)
- feat(event) add lv\_obj\_remove\_event\_cb\_with\_user\_data [4eddeb3](#)
- fix(draw) fix shadow drawing with radius=0 [4250e3c](#)
- fix(msgbox) directly store the pointer of all children [eb5eaa3](#)
- fix(draw) use the filtered colors in lv\_obj\_init\_draw\_xxx\_dsc() functions [78725f2](#)
- fix(arc) fix full arc invalidation [98b9ce5](#)
- chore(led) expose LV\_LED\_BRIGHT\_MIN/MAX in led.h [3f18b23](#)
- fix(group) keep the focused object in lv\_group\_swap\_obj [a997147](#)
- fix(obj) swap objects in the group too in lv\_obj\_swap() [52c7558](#)
- fix(theme) use opacity on button's shadow in the default theme [c5342e9](#)
- fix(win) enable clip\_corner and border\_post by default [493ace3](#)
- fix(draw) fix rectangle drawing with clip\_corner enabled [01237da](#)
- fix(arc) fix other invalidation issues [b0a7337](#)
- feat(obj) add lv\_obj\_get\_x/y\_aligned [98bc1fe](#)
- fix(calendar) fix incorrect highlight of today [adbac52](#)
- fix(arc, meter) fix invalidation in special cases [0f14f49](#)
- fix(canvas) invalidate the image on delete [a1b362c](#)
- fix(msgbox) return the correct pointer from lv\_msgbox\_get\_text [50ea6fb](#)
- fix(bidi) fix the handling of LV\_BASE\_DIR\_AUTO in several widgets [7672847](#)
- fix(build) remove main component dependency (#2420) [f2c2393](#)
- fix(meter) fix inner mask usage [c28c146](#)
- fix(log) fix warning for empty log macros [4dba8df](#)
- fix(theme) improve button focus of keyboard [2504b7e](#)
- fix(tabview) send LV\_EVENT\_VALUE\_CHANGED only once [933d282](#)
- fix(obj style) fix children reposition if the parent's padding changes. [57cf661](#)
- fix(template) update indev template for v8 [d8a3d3d](#)
- fix(obj) detecting which indev sent LV\_EVENT\_FOCUS [f03d4b8](#)
- fix(roller) adjust the size of the selected area correctly [01d1c87](#)
- fix(imgbtn) consider width==LV\_SIZE\_CONTENT if only mid. img is set [7e49f48](#)



- fix(flex) fix NULL pointer dereference [97ba12f](#)
- fix(obj, switch) do not send LV\_EVENT\_VALUE\_CHANGED twice [713b39e](#)
- fix(coords) fix using large coordinates [428db94](#)
- fix(chart) fix crash if no series are added [c728b5c](#)
- fix(meter) fix needle image invalidation [54d8e81](#)
- fix(mem) add lv\_ prefix to tlsf functions and types [0d52b59](#)
- fix(ppx) change LV\_COLOR TRANSP to LV\_COLOR\_CHROMA\_KEY to v8 compatibility [81f3068](#)

## Examples

- example(chart) add area chart example [2507](#)
- example(anim) add demo to use cubic-bezier [2393](#)
- feat(example) add lv\_example\_chart\_9.py [2604](#)
- feat(example) add lv\_example\_chart\_8.py [2611](#)
- feat(example) chart example to add gap between the old and new data [2565](#)
- feat(example) add lv example list 2 [2545](#)
- feat(examples) add MicroPython version of lv\_example\_anim\_3 and allow loading roller font dynamically [2412](#)
- feat(examples) added MP version of second tabview example [2347](#)
- fix(example):format codes [2731](#)
- fix(example) minor fixes in lv\_example\_chart\_2.py [2601](#)
- feat(example) add text with gradient example [462fbcbb](#)
- fix(example\_roller\_3) mask free param bug [2553](#)
- fix(examples) don't compile assets unless needed [2523](#)
- fix(example) scroll example sqort types [2498](#)
- fix(examples) join usage [2425](#)
- fix(examples) add missing lv.PART.INDICATOR [2423](#)
- fix(examples) use lv.grid\_fr for MicroPython [2419](#)
- fix(examples) remove symlinks [2406](#)
- fix(examples) import 'u' -prefixed versions of modules [2365](#)
- fix(examples) remove cast in MP scripts [2354](#)
- fix(examples) fix MicroPython examples and run the examples with CI [2339](#)
- fix(examples) align with renamed Micropython APIs [2338](#)
- fix(examples) adjust canvas example for MicroPython API change [52d1c2e](#)
- fix(example) revert test code [77e2c1f](#)
- feat(example) add checkbox example for radio buttons [d089b36](#)
- feat(example) add text with gradient example [462fbcbb](#)
- fix(examples) exclude example animimg images if animimg is disabled [4d7d306](#)

- fix(example) adjust the object sizes in lv\_example\_anim\_timeline\_1() [71a10e4](#)
- fix(example) revert text code from lv\_example\_checkbox\_2 [28e9593](#)

## Docs

- docs: fix typo [2765](#)
- docs(colorwheel) fix old API names [2643](#)
- docs(display) fix typo [2624](#)
- docs add static for lv\_indev\_drv\_t [2605](#)
- docs(animimg) add to extra widgets index and fix example [2610](#)
- docs(animimg) Add missing animation image page [2609](#)
- docs(group) remove reference to lv\_cont which is gone in v8 [2580](#)
- docs(style) use correct API name for local styles [2550](#)
- docs(all) Proofread, fix typos and add clarifications in confusing areas [2528](#)
- docs(flex) update flex.md [2517](#)
- docs more spelling fixes [2499](#)
- docs fix typo: arae -> area [2488](#)
- docs(readme) fix typo: hosing → hosting. [2477](#)
- docs update company name and year [2476](#)
- docs fix typos [2472](#)
- docs(overview) fix typo [2465](#)
- docs(bar) fix typos in widget examples [2463](#)
- docs(overview) fix typo [2454](#)
- docs(chart) typos [2427](#)
- docs(layout) add internal padding paragraph to grid and flex layout p... [2392](#)
- docs(porting) fix indev example to remove v7 bool return [2381](#)
- docs(README) fix broken references [2329](#)
- docs(grid) typo fix [2310](#)
- docs(color) language fixes [2302](#)
- docs(lv\_obj\_style) update add\_style and remove\_style function headers [2287](#)
- docs(contributing) add commit message format section [3668e54](#)
- docs minor typo fixes [84c0086](#)
- docs(arduino) update some outdated information [9a77102](#)
- docs(keyboard) add note regarding event handler [255f729](#)
- docs minor CSS fix [acbb680](#)
- docs minor CSS improvements [7f367d6](#)
- docs(keyboard) change LV\_KEYBOARD\_MODE\_NUM to LV\_KEYBOARD\_MODE\_NUMBER [6e83d37](#)

- docs(textarea) clarify the use of text selection bg\_color [65673c0](#)
- docs list all examples on one page [25acaf4](#)
- docs(examples) add MicroPython examples [6f37c4f](#)
- docs(filesystem) update to v8 [7971ade](#)
- docs(style) complete the description of style the properties [55e8846](#)
- docs example list fixes [cd600d1](#)
- docs(style) complete the description of style the properties [ff087da](#)
- docs(README) update links, examples, and add services menu [3471bd1](#)
- docs(color) update colors' docs [9056b5e](#)
- docs update lv\_fs.h, layer and align.png to v8 [31ab062](#)
- docs(color) minor fix [ac8f453](#)
- docs update changelog [c386110](#)
- docs(extra) add extra/README.md [8cd504d](#)
- docs add lazy load to the iframes of the examples [c49e830](#)
- docs(os) add example and clarify some points [d996453](#)
- docs(rlottie) fix build error [ce0b564](#)
- docs include paths in libs [f5f9562](#)
- docs libs fixes [8e7bba6](#)
- docs(obj) add comment lv\_obj\_get\_x/y/width/height about postponed layout recalculation [533066e](#)
- docs fix example list [ed77ed1](#)
- docs describe the options to include or skip lv\_conf.h [174ef66](#)
- docs(overview) spelling fixes [d2efb8c](#)
- docs(table) describe keypad/encoder navigation [749d1b3](#)
- docs update CHANGELOG [0f8bc18](#)
- docs(image) mention the frame\_id paramter of lv\_img\_decoder\_open [2433732](#)
- docs(arduino) update how to use the examples [06962a5](#)
- docs(rlottie): fix typo in commands [ed9169c](#)
- docs(indev, layer) update lv\_obj\_set\_click() to lv\_obj\_add\_flag() [bcd99e8](#)
- docs update version support table [e6e98ab](#)
- docs fix example list [c6f99ad](#)
- docs(examples) add <hr/> to better separate examples [a1b59e3](#)
- docs(checkbox) update the comment lv\_checkbox\_set\_text\_static [3e0ddd0](#)
- docs(grid) fix missing article [da0c97a](#)
- docs(display) fix grammar in one spot [5dbea7d](#)
- docs(style) fix typo in style property descriptions [4e3b860](#)
- docs(flex) fix typo in flex grow section [e5fafc4](#)

- docs(indev) clarify purpose of `continue_reading` flag [706f81e](#)
- docs(license) update company name and year [7c1eb00](#)
- docs fix typo [8ab8064](#)
- docs add libs to the main index [1a8fed5](#)
- docs add `btn_example.png` [8731ef1](#)
- docs(btnmatrix) fix typo with `set_all/clear_all` parameters [51a82a1](#)

## CI and tests

- ci(micropython) fix git fetch [2757](#)
- test(txt) initial unit tests and general code cleanup/fixes [2623](#)
- test add `setUp` and `tearDown` to test template [2648](#)
- test(arc) add initial unit tests [2617](#)
- ci(micropython) add ESP32 and STM32 tests [2629](#)
- test(checkbox) add initial tests [2551](#)
- test(ci) build and run tests in parallel. [2515](#)
- ci(tests) run tests using `ctest` [2503](#)
- ci(tests) add dependency on GNU parallel [2510](#)
- ci(tests) use common script to install development prereqs [2504](#)
- test convert Makefile to CMake [2495](#)
- test Refactor unit test scripts. [2473](#)
- test(font\_loader) migrate the existing font loader test [bc5b3be](#)
- test add build test again, add dropdown test, integrate gcov and gvoctr [e35b1d0](#)
- test(dropdown) add test for keypad and encoder [4143b80](#)
- test add keypad and encoder emulators [e536bb6](#)
- tests add mosue emulator [2ba810b](#)
- tests add README [b765643](#)
- test add move tests to `test_cases` and `test_runners` directories [e9e010a](#)
- test fix CI build error [c38cae2](#)
- ci add config for 8bpp [3eacc59](#)
- test move more source files to `src` folder [3672f87](#)
- test update CI for the new tests [a3898b9](#)
- test clean up report folder [b9b4ba5](#)
- test fix build error [61cda59](#)
- test(font\_loader) migrate the existing font loader test [d6dbbaa](#)
- test add move tests to `test_cases` and `test_runners` directories [d2e735e](#)
- test add 3rd party libs to all tests and also fix them [7a95fa9](#)

- test(arc): add test case for adv\_hittest [e83df6f](#)
- ci create check for lv\_conf\_internal.h [5d8285e](#)
- test fix warning and docs build error [d908f31](#)
- ci(micropython) add rp2 port [1ab5c96](#)
- test(dropdown) remove dummy test case [9fb98da](#)
- ci(codecov) hide statuses on commits for now [0b7be77](#)
- ci(docs) run apt-get update before installation [f215174](#)
- test fix LV\_USE\_LOG\_LEVEL -> LV\_LOG\_LEVEL typo [80f0b09](#)
- ci(micropython) add GCC problem matcher [ab316a0](#)
- test convert Makefile to CMake (#2495) [9c846ee](#)

## Others

- chore: replace (void)xxx with LV\_UNUSED(xxx) [2779](#)
- animation improvement [2743](#)
- Improve LV\_FORMAT\_ATTRIBUTE usage [2673](#)
- Fix typo in commands to build rlottie [2723](#)
- del(.gitmodules): delete .gitmodules [2718](#)
- lv\_obj\_draw\_part\_dsc\_t.text\_length added [2694](#)
- expose LV\_COLOR\_DEPTH and LV\_COLOR\_16\_SWAP in micropython [2679](#)
- sync lvgl/lv\_fs\_if [2676](#)
- build: always enable CMake install rule in default configuration [2636](#)
- build: fix lib name in CMakeLists [2641](#)
- build: remove use of ‘project’ keyword in CMakeLists [2640](#)
- build add install rule to CMakeList.txt [2621](#)
- Fixed row size calculation [2633](#)
- arch add small 3rd party libs to lvgl [2569](#)
- Kconfig: Add missing options [2597](#)
- Espressif IDF component manager [2521](#)
- chore(btnmatrix) removed unnecessary semicolon [2520](#)
- Update README.md [2516](#)
- Corrected a function name in obj.md [2511](#)
- Simple spelling fixes [2496](#)
- added lv\_obj\_move\_up() and lv\_obj\_move\_down() [2467](#)
- Fix buf name error for “lv\_port\_disp\_template.c” and optimize the arduino example [2475](#)
- Fix two examples in the docs with new v8 api [2486](#)
- kconfig: minor fix for default dark theme option [2426](#)

- doc(table) update doc on cell merging [2397](#)
- added example lv\_example\_anim\_timeline\_1.py [2387](#)
- refactor(sprintf) add printf-like function attribute to \_lv\_txt\_set\_text\_vfmt and lv\_label\_set\_text\_fmt [2332](#)
- Update win.md [2352](#)
- Nxp pxp vglite v8 dev [2313](#)
- More Snapable → Snappable replacements [2304](#)
- Spelling and other language fixes to documentation [2293](#)
- Update quick-overview.md [2295](#)
- adding micropython examples [2286](#)
- format run code-formatter.sh [d67dd94](#)
- Update ROADMAP.md [2b1ae3c](#)
- Create .codecov.yml [e53aa82](#)
- refactor(examples) drop JS-specific code from header.py [ef41450](#)
- make test run on mseter and release/v8.\* [227402a](#)
- Update release.yml [0838f12](#)
- refactor(examples) drop usys import from header.py [ad1f91a](#)
- Update ROADMAP.md [a38fcf2](#)
- Revert “feat(conf) add better check for Kconfig default” [a5793c7](#)
- remove temporary test file [a958c29](#)
- start to implement release/patch [1626a0c](#)
- chore(indev) minor formatting [79ab3d2](#)
- add basic patch release script [1c3ecf1](#)
- chore(example) minor improvements on lv\_example\_list\_2 [bb6d6b7](#)
- tool: add changelog\_gen.sh to automatically generate changelog [6d95521](#)
- update version numbers to v8.1.0-dev [8691611](#)
- chore(test) improve prints [ea8bed3](#)
- chore(test) improve prints [0c4bca0](#)
- chore: update lv\_conf\_internal.h [41c2dd1](#)
- chore(format) lv\_conf\_template.h minor formatting [3c86d77](#)
- chore(docs) always deploy master to docs/master as well [6d05692](#)
- Update CHANGELOG.md [48fd73d](#)
- Fix compile errors [6c956cc](#)
- Update textarea.md [6d8799f](#)
- chore(assert) add warnign about higher memory usage if LV\_USE\_ASSERT\_STYLE is enabled [33e4330](#)
- Update page.html [9573bab](#)
- chore(docs) force docs rebuild [4a0f413](#)

- Fix typo error in color.md [572880c](#)
- Update arc.md [2a9b9e6](#)
- Update index.rst [9ce2c77](#)
- chore(docs) minor formatting on example' s GitHub link [75209e8](#)
- chore(lv\_conf\_template) fix spelling mistake [9d134a9](#)
- Update CHANGELOG.md [8472360](#)
- chore(stale) disable on forks [93c1303](#)
- Revert “fix(tests) remove src/test\_runners when cleaning” [ae15a1b](#)
- style fix usage of clang-format directives [2122583](#)
- Revert “fix(indev) focus on objects on release instead of press” [f61b2ca](#)

### 1.10.2 v8.0.2 (16.07.2021)

- fix(theme) improve button focus of keyboard
- fix(tabview) send LV\_EVENT\_VALUE\_CHANGED only once
- fix(imgbtn) use the correct src in LV\_EVENT\_GET\_SELF\_SIZE
- fix(color) remove extraneous cast for 8-bit color
- fix(obj style) fix children reposition if the parent' s padding changes.
- fix(color) remove extraneous \_LV\_COLOR\_MAKE\_TYPE\_HELPER (#2372)
- fix(spinner) should not be clickable (#2373)
- fix(obj) improve how the focusing indev is determined
- fix(template) update indev template for v8
- fix(sprintf) skip defining attribute if pycparser is used
- refactor(sprintf) add printf-like function attribute to \_lv\_txt\_set\_text\_vfmt and lv\_label\_set\_text\_fmt (#2332)
- fix(template) include lvgl.h in lv\_port\_\*\_template.c files
- fix(obj) detecting which indev sent LV\_EVENT\_FOCUS
- fix (span) fill LV\_EVENT\_GET\_SELF\_SIZE (#2360)
- fix(arc) disable LV\_OBJ\_FLAG\_SCROLL\_CHAIN by default
- fix (draw) fix arc bg image drawing with full arcs
- fix(disp) fix memory leak in lv\_disp\_remove (#2355)
- fix warnigs introduced by 3fb8baf5
- fix(widgets) use lv\_obj\_class for all the widgets
- fix(obj) move clean ups from lv\_obj\_del to lv\_obj\_destructor
- fix(roller) fix partial redraw of the selected area
- fix(roller) adjust the size of the selected area correctly
- fix(obj) delete useless type conversion (#2343)
- fix(lv\_obj\_scroll.h) typos (#2345)

- fix(scroll) fire LV\_EVENT\_SCROLL\_BEGIN in the same spot for both axes
- fix(btnmatrix) fix button invalidation on focus change
- fix(textarea) style update in oneline mode + improve scroll to cursor
- fix(tlsf) do not use <assert.h>
- fix(imgbtn) consider width==LV\_SIZE\_CONTENT if only mid. img is set
- fix(refr) reduce the nesting level in lv\_refr\_area
- fix(txt) enhance the function of break\_chars (#2327)
- fix(pxp): update RTOS macro for SDK 2.10
- fix(vglite): update for v8
- fix(pxp): update for v8
- fix(flex) fix layout update and invalidation issues
- fix(flex) fix NULL pointer dereference
- fix(obj, switch) do not send LV\_EVENT\_VALUE\_CHANGED twice
- fix(color) overflow with 16-bit color depth
- fix(coords) fix using large coordinates
- fix(chart) fix crash if no series are added
- fix(chart) invalidation with LV\_CHART\_UPDATE\_MODE\_SHIFT
- fix(align) fix lv\_obj\_align\_to G
- fix(table) invalidate the table on cell value change
- fix(label) remove duplicated lv\_obj\_refresh\_self\_size
- fix(draw) underflow in subpixel font drawing
- fix (scroll) do not send unnecessary scroll end events

### 1.10.3 v8.0.1 (14.06.2021)

- docs(filesystem) update to v8 7971ade4
- fix(msgbox) create modals on top layer instead of act screen 5cf6303e
- fix(colorwheel) disable LV\_OBJ\_FLAG\_SCROLL\_CHAIN by default 48d1c292
- docs(grid) typo fix (#2310) 69d109d2
- fix(arduino) fix the prototype of my\_touchpad\_read in the LVGL\_Arduino.ino 1a62f7a6
- fix(meter) fix needle image invalidation 54d8e817
- fix(mem) add lv\_ prefix to tlsf functions and types 0d52b59c
- fix(calendar) fix the position calculation today ad05e196
- fix(typo) rename LV\_OBJ\_FLAG\_SNAPABLE to LV\_OBJ\_FLAG\_SNAPPABLE e697807c
- docs(color) language fixes (#2302) 07ecc9f1
- fix(tick) minor optimization on lv\_tick\_inc call test b4305df5
- Spelling and other language fixes to documentation (#2293) d0aaacaf



- fix(theme) show disabled state on buttons of btnmatrix, msgbox and keyboard 0be582b3
- fix(scroll) keep the scroll position on object deleted 52edbb46
- fix(msgbox) handle NULL btn map parameter 769c4a30
- fix(group) allow refocusing objects 1520208b
- docs(overview) spelling fixes d2efb8c6
- Merge branch 'master' of <https://github.com/lvgl/lvgl> 45960838
- feat(timer) check if lv\_tick\_inc is called aa6641a6
- feat(docs) add view on GitHub link a716ac6e
- fix(theme) fix the switch style in the default theme 0c0dc8ea
- docs fix typo 8ab80645
- Merge branch 'master' of <https://github.com/lvgl/lvgl> e796448f
- feat(event) pass the scroll animation to LV\_EVENT\_SCROLL\_BEGIN ca54ecfe
- fix(tabview) fix with left and right tabs 17c57449
- chore(docs) force docs rebuild 4a0f4139
- chore(docs) always deploy master to docs/master as well 6d05692d
- fix(template) update lv\_objx\_template to v8 38bb8afc
- docs(extra) add extra/README.md 8cd504d5
- Update CHANGELOG.md 48fd73d2
- Update quick-overview.md (#2295) 5616471c
- fix(pxp) change LV\_COLOR\_TRANSP to LV\_COLOR\_CHROMA\_KEY to v8 compatibility 81f3068d
- adding micropython examples (#2286) c60ed68e
- docs(color) minor fix ac8f4534
- fix(example) revert test code 77e2c1ff
- fix(draw) with additive blending with 32-bit color depth 786db2af
- docs(color) update colors' docs 9056b5ee
- Merge branch 'master' of <https://github.com/lvgl/lvgl> a711a1dd
- perf(refresh) optimize where to wait for lv\_disp\_flush\_ready with 2 buffers d0172f14
- docs(lv\_obj\_style) update add\_style and remove\_style function headers (#2287) 60f7bcbf
- fix memory leak of spangroup (#2285) 33e0926a
- fix make lv\_img\_cache.h public because cache invalidation is public 38ebcd81
- Merge branch 'master' of <https://github.com/lvgl/lvgl> 2b292495
- fix(btnmatrix) fix focus event handling 3b58ef14
- Merge pull request #2280 from lvgl/dependabot/pip/docs/urllib3-1.26.5 a2f45b26
- fix(label) calculating the clip area 57e211cc
- chore(deps): bump urllib3 from 1.26.4 to 1.26.5 in /docs b2f77dfc
- fix(docs) add docs about the default group 29bfe604

### 1.10.4 v8.0.0 (01.06.2021)

v8.0 brings many new features like simplified and more powerful scrolling, new layouts inspired by CSS Flexbox and Grid, simplified and improved widgets, more powerful events, hookable drawing, and more.

v8 is a major change and therefore it's not backward compatible with v7.

#### Directory structure

- The `lv_` prefix is removed from the folder names
- The `docs` is moved to the `lvgl` repository
- The `examples` are moved to the `lvgl` repository
- Create an `src/extra` folder for complex widgets:
  - It makes the core LVGL leaner
  - In `extra` we can have a lot and specific widgets
  - Good place for contributions

#### Widget changes

- `lv_cont` removed, layout features are moved to `lv_obj`
- `lv_page` removed, scroll features are moved to `lv_obj`
- `lv_objmask` the same can be achieved by events
- `lv_meter` added as the union of `lv_linemeter` and `lv_gauge`
- `lv_span` new widget mimicking HTML `<span>`
- `lv_animing` new widget for simple slideshow animations
- + many minor changes and improvements

#### New scrolling

- Support “elastic” scrolling when scrolled in
- Support scroll chaining among any objects types (not only `lv_page`s)
- Remove `lv_drag`. Similar effect can be achieved by setting the position in `LV_EVENT_PRESSING`
- Add snapping
- Add snap stop to scroll max 1 snap point

## New layouts

- CSS Grid-like layout support
- CSS Flexbox-like layout support

## Styles

- Optimize and simplify styles
- State is saved in the object instead of the style property
- Object size and position can be set in styles too

## Events

- Allow adding multiple events to an object
- A `user_data` can be attached to the added events

## Driver changes

- `lv_disp_drv_t`, `lv_indev_drv_t`, `lv_fs_drv_t` needs to be `static`
- `...disp_buf...` is renamed to `draw_buf`. See an initialization example [here](#).
- No partial update if two screen sized buffers are set
- `disp_drv->full_refresh = 1` makes always the whole display redraw.
- `hor_res` and `ver_res` need to be set in `disp_drv`
- `indev_read_cb` returns `void`. To indicate that there is more that to read set `data->continue_reading = 1` in the `read_cb`

## Other changes

- Remove the copy parameter from create functions
- Simplified File system interface API
- Use a more generic inheritance
- The built-in themes are reworked
- `lv_obj_align` now saved the alignment and realigns the object automatically but can't be used to align to other than the parent
- `lv_obj_align_to` can align to an object but doesn't save the alignment
- `lv_pct(x)` can be used to set the size and position in percentage
- There are many other changes in widgets that are not detailed here. Please refer to the documentation of the widgets.

## New release policy

- We will follow [Release branches with GitLab flow](#)
- Minor releases are expected in every 3-4 month
- `master` will always contain the latest changes

## Migrating from v7 to v8

- First and foremost, create a new `lv_conf.h` based on `lv_conf_template.h`.
- To try the new version it's recommended to use a simulator project and see the examples.
- When migrating your project to v8
  - Update the drivers are described above
  - Update the styles
  - Update the events
  - Use the new layouts instead of `lv_cont` features
  - Use `lv_obj` instead of `lv_page`
  - See the changes in [Colors](#)
  - The other parts are mainly minor renames and refactoring. See the functions' documentation for descriptions.

### 1.10.5 v7.11.0 (16.03.2021)

#### New features

- Add better screen orientation management with software rotation support
- Decide text animation's direction based on `base_dir` (when using `LV_USE_BIDI`)

#### Bugfixes

- `fix(gauge)` fix needle invalidation
- `fix(bar)` correct symmetric handling for vertical sliders

### 1.10.6 v7.10.1 (16.02.2021)

#### Bugfixes

- `fix(draw)` overlap outline with background to prevent aliasing artifacts
- `fix(indev)` clear the `indev`'s `act_obj` in `lv_indev_reset`
- `fix(text)` fix out of bounds read in `_lv_txt_get_width`
- `fix(list)` scroll list when button is focused using `LV_KEY_NEXT/PREV`
- `fix(text)` improve Arabic contextual analysis by adding hyphen processing and proper handling of lam-alef sequence
- `fix(delete)` delete animation after the children are deleted

- fix(gauge) consider paddings for needle images

### 1.10.7 v7.10.0 (02.02.2021)

#### New features

- feat(indev) allow input events to be passed to disabled objects
- feat(spinbox) add inline get\_step function for MicroPython support

#### Bugfixes

- fix(btnmatrix) fix lv\_btnmatrix\_get\_active\_btn\_text() when used in a group

### 1.10.8 v7.9.1 (19.01.2021)

#### Bugfixes

- fix(cpicker) fix division by zero
- fix(dropdown) fix selecting options after the last one
- fix(msgbox) use the animation time provided
- fix(gpu\_nxp\_pxp) fix incorrect define name
- fix(indev) don't leave edit mode if there is only one object in the group
- fix(draw\_rect) fix draw pattern stack-use-after-scope error

### 1.10.9 v7.9.0 (05.01.2021)

#### New features

- feat(chart) add lv\_chart\_remove\_series and lv\_chart\_hide\_series
- feat(img\_cache) allow disabling image caching
- calendar: make get\_day\_of\_week() public
- Added support for Zephyr integration

#### Bugfixes

- fix(draw\_rect) free buffer used for arabic processing
- fix(win) arabic process the title of the window
- fix(dropdown) arabic process the option in lv\_dropdown\_add\_option
- fix(textarea) buffer overflow in password mode with UTF-8 characters
- fix(textarea) cursor position after hiding character in password mode
- fix(linewriter) draw critical lines with correct color

- fix(lv\_conf\_internal) be sure Kconfig defines are always uppercase
- fix(kconfig) handle disable sprintf float correctly.
- fix(layout) stop layout after recursion threshold is reached
- fix(gauge) fix redraw with image needle

### 1.10.10 v7.8.1 (15.12.2020)

#### Bugfixes

- fix(lv\_scr\_load\_anim) fix when multiple screens are loaded at the same time with delay
- fix(page) fix LV\_SCROLLBAR\_MODE\_DRAG

### 1.10.11 v7.8.0 (01.12.2020)

#### New features

- make DMA2D non blocking
- add unscii-16 built-in font
- add KConfig
- add lv\_refr\_get\_fps\_avg()

#### Bugfixes

- fix(btnmatrix) handle arabic texts in button matrices
- fix(indev) disabled object shouldn't absorb clicks but let the parent to be clicked
- fix(arabic) support processing again already processed texts with \_lv\_txt\_ap\_proc
- fix(textarea) support Arabic letter connections
- fix(dropdown) support Arabic letter connections
- fix(value\_str) support Arabic letter connections in value string property
- fix(indev) in LV\_INDEV\_TYPE\_BUTTON recognize 1 cycle long presses too
- fix(arc) make arc work with encoder
- fix(slider) adjusting the left knob too with encoder
- fix reference to LV\_DRAW\_BUF\_MAX\_NUM in lv\_mem.c
- fix(polygon draw) join adjacent points if they are on the same coordinate
- fix(linometer) fix invalidation when setting new value
- fix(table) add missing invalidation when changing cell type
- refactor(roller) rename LV\_ROLLER\_MODE\_INIFINITE -> LV\_ROLLER\_MODE\_INFINITE

### 1.10.12 v7.7.2 (17.11.2020)

#### Bugfixes

- fix(draw\_triangle): fix polygon/triangle drawing when the order of points is counter-clockwise
- fix(btnmatrix): fix setting the same map with modified pointers
- fix(arc) fix and improve arc dragging
- label: Repair calculate back `dot` character logical error which cause infinite loop.
- fix(theme\_material): remove the bottom border from tabview header
- fix(imgbtn) guess the closest available state with valid src
- fix(spinbox) update cursor position in lv\_spinbox\_set\_step

### 1.10.13 v7.7.1 (03.11.2020)

#### Bugfixes

- Respect btnmatrix's `one_check` in `lv_btnmatrix_set_btn_ctrl`
- Gauge: make the needle images to use the styles from `LV_GAUGE_PART_PART`
- Group: fix in `lv_group_remove_obj` to handle deleting hidden objects correctly

### 1.10.14 v7.7.0 (20.10.2020)

#### New features

- Add PXP GPU support (for NXP MCUs)
- Add VG-Lite GPU support (for NXP MCUs)
- Allow max. 16 cell types for table
- Add `lv_table_set_text_fmt()`
- Use margin on calendar header to set distances and padding to the size of the header
- Add `text_sel_bg` style property

#### Bugfixes

- Theme update to support text selection background
- Fix imgbtn state change
- Support RTL in table (draw columns right to left)
- Support RTL in pretty layout (draw columns right to left)
- Skip objects in groups if they are in disabled state
- Fix dropdown selection with RTL basedirection
- Fix rectangle border drawing with large width
- Fix `lv_win_clean()`

### 1.10.15 v7.6.1 (06.10.2020)

#### Bugfixes

- Fix BIDI support in dropdown list
- Fix copying base dir in `lv_obj_create`
- Handle sub pixel rendering in font loader
- Fix transitions with style caching
- Fix click focus
- Fix imgbtn image switching with empty style
- Material theme: do not set the text font to allow easy global font change

### 1.10.16 v7.6.0 (22.09.2020)

#### New features

- Check whether any style property has changed on a state change to decide if any redraw is required

#### Bugfixes

- Fix selection of options with non-ASCII letters in dropdown list
- Fix font loader to support `LV_FONT_FMT_TXT_LARGE`

### 1.10.17 v7.5.0 (15.09.2020)

#### New features

- Add `clean_dcache_cb` and `lv_disp_clean_dcache` to enable users to use their own cache management function
- Add `gpu_wait_cb` to wait until the GPU is working. It allows to run CPU a wait only when the rendered data is needed.
- Add 10px and 8ox built in fonts

#### Bugfixes

- Fix unexpected `DEFOCUS` on `lv_page` when clicking to bg after the scrollable
- Fix `lv_obj_del` and `lv_obj_clean` if the children list changed during deletion.
- Adjust button matrix button width to include padding when spanning multiple units.
- Add rounding to btnmatrix line height calculation
- Add `decmodpr_buf` to GC roots
- Fix division by zero in `draw_pattern` (`lv_draw_rect.c`) if the image or letter is not found
- Fix drawing images with 1 px height or width



### 1.10.18 v7.4.0 (01.09.2020)

The main new features of v7.4 are run-time font loading, style caching and arc knob with value setting by click.

#### New features

- Add `lv_font_load()` function - Loads a `lv_font_t` object from a binary font file
- Add `lv_font_free()` function - Frees the memory allocated by the `lv_font_load()` function
- Add style caching to reduce access time of properties with default value
- arc: add set value by click feature
- arc: add `LV_ARC_PART_KNOB` similarly to slider
- send gestures event if the object was dragged. User can check dragging with `lv_indev_is_dragging(lv_indev_act())` in the event function.

#### Bugfixes

- Fix color bleeding on border drawing
- Fix using 'LV\_SCROLLBAR\_UNHIDE' after 'LV\_SCROLLBAR\_ON'
- Fix cropping of last column/row if an image is zoomed
- Fix zooming and rotating mosaic images
- Fix deleting tabview with LEFT/RIGHT tab position
- Fix btnmatrix to not send event when `CLICK_TRIG` = true and the cursor slid from a pressed button
- Fix roller width if selected text is larger than the normal

### 1.10.19 v7.3.1 (18.08.2020)

#### Bugfixes

- Fix drawing value string twice
- Rename `lv_chart_clear_serie` to `lv_chart_clear_series` and `lv_obj_align_origo` to `lv_obj_align_mid`
- Add linemeter's mirror feature again
- Fix text decor (underline strikethrough) with older versions of font converter
- Fix setting local style property multiple times
- Add missing background drawing and radius handling to image button
- Allow adding extra label to list buttons
- Fix crash if `lv_table_set_col_cnt` is called before `lv_table_set_row_cnt` for the first time
- Fix overflow in large image transformations
- Limit extra button click area of button matrix's buttons. With large paddings it was counter-intuitive. (Gaps are mapped to button when clicked).
- Fix `lv_btnmatrix_set_one_check` not forcing exactly one button to be checked

- Fix color picker invalidation in rectangle mode
- Init disabled days to gray color in calendar

### 1.10.20 v7.3.0 (04.08.2020)

#### New features

- Add `lv_task_get_next`
- Add `lv_event_send_refresh`, `lv_event_send_refresh_recursive` to easily send `LV_EVENT_REFRESH` to object
- Add `lv_tabview_set_tab_name()` function - used to change a tab's name
- Add `LV_THEME_MATERIAL_FLAG_NO_TRANSITION` and `LV_THEME_MATERIAL_FLAG_NO_FOCUS` flags
- Reduce code size by adding: `LV_USE_FONT_COMPRESSED` and `LV_FONT_USE_SUBPX` and applying some optimization
- Add `LV_MEMCPY_MEMSET_STD` to use standard `memcpy` and `memset`

#### Bugfixes

- Do not print warning for missing glyph if its height OR width is zero.
- Prevent duplicated sending of `LV_EVENT_INSERT` from text area
- Tidy outer edges of cpicker widget.
- Remove duplicated lines from `lv_tabview_add_tab`
- `btnmatrix`: handle combined states of buttons (e.g. checked + disabled)
- `textarea`: fix typo in `lv_textarea_set_scrollbar_mode`
- `gauge`: fix image needle drawing
- fix using freed memory in `_lv_style_list_remove_style`

### 1.10.21 v7.2.0 (21.07.2020)

#### New features

- Add screen transitions with `lv_scr_load_anim()`
- Add display background color, wallpaper and opacity. Shown when the screen is transparent. Can be used with `lv_disp_set_bg_opa/color/image()`.
- Add `LV_CALENDAR_WEEK_STARTS_MONDAY`
- Add `lv_chart_set_x_start_point()` function - Set the index of the x-axis start point in the data array
- Add `lv_chart_set_ext_array()` function - Set an external array of data points to use for the chart
- Add `lv_chart_set_point_id()` function - Set an individual point value in the chart series directly based on index

- Add `lv_chart_get_x_start_point()` function - Get the current index of the x-axis start point in the data array
- Add `lv_chart_get_point_id()` function - Get an individual point value in the chart series directly based on index
- Add `ext_buf_assigned` bit field to `lv_chart_series_t` structure - it's true if external buffer is assigned to series
- Add `lv_chart_set_series_axis()` to assign series to primary or secondary axis
- Add `lv_chart_set_y_range()` to allow setting range of secondary y-axis (based on `lv_chart_set_range` but extended with an axis parameter)
- Allow setting different font for the selected text in `lv_roller`
- Add `theme->apply_cb` to replace `theme->apply_xcb` to make it compatible with the MicroPython binding
- Add `lv_theme_set_base()` to allow easy extension of built-in (or any) themes
- Add `lv_obj_align_x()` and `lv_obj_align_y()` functions
- Add `lv_obj_align_origo_x()` and `lv_obj_align_origo_y()` functions

### Bugfixes

- `tileview` fix navigation when not screen sized
- Use 14px font by default to for better compatibility with smaller displays
- `linemeter` fix conversation of current value to "level"
- Fix drawing on right border
- Set the cursor image non-clickable by default
- Improve mono theme when used with keyboard or encoder

## 1.10.22 v7.1.0 (07.07.2020)

### New features

- Add `focus_parent` attribute to `lv_obj`
- Allow using buttons in encoder input device
- Add `lv_btnmatrix_set/get_align` capability
- DMA2D: Remove dependency on ST CubeMX HAL
- Added `max_used` propriety to `lv_mem_monitor_t` struct
- In `lv_init` test if the strings are UTF-8 encoded.
- Add `user_data` to themes
- Add `LV_BIG_ENDIAN_SYSTEM` flag to `lv_conf.h` in order to fix displaying images on big endian systems.
- Add inline function `lv_checkbox_get_state(const lv_obj_t * cb)` to extend the checkbox functionality.
- Add inline function `lv_checkbox_set_state(const lv_obj_t * cb, lv_btn_state_t state)` to extend the checkbox functionality.

## Bugfixes

- `lv_img` fix invalidation area when angle or zoom changes
- Update the style handling to support Big endian MCUs
- Change some methods to support big endian hardware.
- remove use of c++ keyword 'new' in parameter of function `lv_theme_set_base()`.
- Add `LV_BIG_ENDIAN_SYSTEM` flag to `lv_conf.h` in order to fix displaying images on big endian systems.
- Fix inserting chars in text area in big endian hardware.

### 1.10.23 v7.0.2 (16.06.2020)

## Bugfixes

- `lv_textarea` fix wrong cursor position when clicked after the last character
- Change all text related indices from 16-bit to 32-bit integers throughout whole library. #1545
- Fix gestures
- Do not call `set_px_cb` for transparent pixel
- Fix list button focus in material theme
- Fix crash when a text area is cleared with the backspace of a keyboard
- Add version number to `lv_conf_template.h`
- Add log in true double buffering mode with `set_px_cb`
- `lv_dropdown`: fix missing `LV_EVENT_VALUE_CHANGED` event when used with encoder
- `lv_tileview`: fix if not the {0;0} tile is created first
- `lv_debug`: restructure to allow asserting in from `lv_misc` too
- add assert if `_lv_mem_buf_get()` fails
- `lv_textarea`: fix character delete in password mode
- Update `LV_OPA_MIN` and `LV_OPA_MAX` to widen the opacity processed range
- `lv_btnm` fix sending events for hidden buttons
- `lv_gaguge` make `lv_gauge_set_angle_offset` offset the labels and needles too
- Fix typo in the API `scrlable -> scrollable`
- `tabview` by default allow auto expanding the page only to right and bottom (#1573)
- fix crash when drawing gradient to the same color
- chart: fix memory leak
- `img`: improve hit test for transformed images

### 1.10.24 v7.0.1 (01.06.2020)

#### Bugfixes

- Make Micropython working by adding the required variables as GC\_ROOT
- Prefix some internal API functions with \_ to reduce the API of LVGL
- Fix built-in SimSun CJK font
- Fix UTF-8 encoding when LV\_USE\_ARABIC\_PERSIAN\_CHARS is enabled
- Fix DMA2D usage when 32 bit images directly blended
- Fix lv\_roller in infinite mode when used with encoder
- Add lv\_theme\_get\_color\_secondary()
- Add LV\_COLOR\_MIX\_ROUND\_OFS to adjust color mixing to make it compatible with the GPU
- Improve DMA2D blending
- Remove memcpy from lv\_ll (caused issues with some optimization settings)
- lv\_chart fix X tick drawing
- Fix vertical dashed line drawing
- Some additional minor fixes and formattings

### 1.10.25 v7.0.0 (18.05.2020)

#### Documentation

The docs for v7 is available at <https://docs.littlevgl.com/v7/en/html/index.html>

#### Legal changes

The name of the project is changed to LVGL and the new website is on <https://lvgl.io>

LVGL remains free under the same conditions (MIT license) and a company is created to manage LVGL and offer services.

#### New drawing system

Complete rework of LVGL's draw engine to use “masks” for more advanced and higher quality graphical effects. A possible use-case of this system is to remove the overflowing content from the rounded edges. It also allows drawing perfectly anti-aliased circles, lines, and arcs. Internally, the drawings happen by defining masks (such as rounded rectangle, line, angle). When something is drawn the currently active masks can make some pixels transparent. For example, rectangle borders are drawn by using 2 rectangle masks: one mask removes the inner part and another the outer part.

The API in this regard remained the same but some new functions were added:

- lv\_img\_set\_zoom: set image object's zoom factor
- lv\_img\_set\_angle: set image object's angle without using canvas
- lv\_img\_set\_pivot: set the pivot point of rotation

The new drawing engine brought new drawing features too. They are highlighted in the “style” section.

## New style system

The old style system is replaced with a new more flexible and lightweight one. It uses an approach similar to CSS: support cascading styles, inheriting properties and local style properties per object. As part of these updates, a lot of objects were reworked and the APIs have been changed.

- more shadows options: *offset* and *spread*
- gradient stop position to shift the gradient area and horizontal gradient
- LV\_BLEND\_MODE\_NORMAL/ADDITIVE/SUBTRACTIVE blending modes
- *clip corner*: crop the content on the rounded corners
- *text underline* and *strikethrough*
- dashed vertical and horizontal lines (*dash gap*, *dash width*)
- *outline*: a border-like part drawn out of the background. Can have spacing to the background.
- *pattern*: display an image in the middle of the background or repeat it
- *value* display a text which is stored in the style. It can be used e.g. as a light-weighted text on buttons too.
- *margin*: similar to *padding* but used to keep space outside the object

Read the [Style](#) section of the documentation to learn how the new styles system works.

## GPU integration

To better utilize GPUs, from this version GPU usage can be integrated into LVGL. In `lv_conf.h` any supported GPUs can be enabled with a single configuration option.

Right now, only ST's DMA2D (Chrom-ART) is integrated. More will in the upcoming releases.

## Renames

The following object types are renamed:

- sw -> switch
- ta -> textarea
- cb -> checkbox
- lmeter -> linemeter
- mbox -> msgbox
- ddlist -> dropdown
- btnm -> btnmatrix
- kb -> keyboard
- preload -> spinner
- lv\_objx folder -> lv\_widgets
- LV\_FIT\_FILL -> LV\_FIT\_PARENT
- LV\_FIT\_FLOOD -> LV\_FLOOD\_MAX
- LV\_LAYOUT\_COL\_L/M/R -> LV\_LAYOUT\_COLUMN\_LEFT/MID/RIGHT

- LV\_LAYOUT\_ROW\_T/M/B -> LV\_LAYOUT\_ROW\_TOP/MID/BOTTOM

### Reworked and improved object

- **dropdown**: Completely reworked. Now creates a separate list when opened and can be dropped to down/up/left/right.
- **label**: `body_draw` is removed, instead, if its style has a visible background/border/shadow etc it will be drawn. Padding really makes the object larger (not just virtually as before)
- **arc**: can draw background too.
- **btn**: doesn't store styles for each state because it's done naturally in the new style system.
- **calendar**: highlight the pressed datum. The used styles are changed: use `LV_CALENDAR_PART_DATE` normal for normal dates, checked for highlighted, focused for today, pressed for the being pressed. (checked+pressed, focused+pressed also work)
- **chart**: only has `LINE` and `COLUMN` types because with new styles all the others can be described. `LV_CHART_PART_SERIES` sets the style of the series. `bg_opa > 0` draws an area in `LINE` mode. `LV_CHART_PART_SERIES_BG` also added to set a different style for the series area. Padding in `LV_CHART_PART_BG` makes the series area smaller, and it ensures space for axis labels/numbers.
- **linemeter, gauge**: can have background if the related style properties are set. Padding makes the scale/lines smaller. `scale_border_width` and `scale_end_border_width` allow to draw an arc on the outer part of the scale lines.
- **gauge**: `lv_gauge_set_needle_img` allows use image as needle
- **canvas**: allow drawing to true color alpha and alpha only canvas, add `lv_canvas_blur_hor/ver` and rename `lv_canvas_rotate` to `lv_canvas_transform`
- **textarea**: If available in the font use bullet (U+2022) character in text area password

### New object types

- **lv\_objmask**: masks can be added to it. The children will be masked accordingly.

### Others

- Change the built-in fonts to `Montserrat` and add built-in fonts from 12 px to 48 px for every 2nd size.
- Add example CJK and Arabic/Persian/Hebrew built-in font
- Add ° and “bullet” to the built-in fonts
- Add Arabic/Persian script support: change the character according to its position in the text.
- Add `playback_time` to animations.
- Add `repeat_count` to animations instead of the current “repeat forever” .
- Replace `LV_LAYOUT_PRETTY` with `LV_LAYOUT_PRETTY_TOP/MID/BOTTOM`

## Demos

- `lv_examples` was reworked and new examples and demos were added

## New release policy

- Maintain this Changelog for every release
- Save old major version in new branches. E.g. `release/v6`
- Merge new features and fixes directly into `master` and release a patch or minor releases every 2 weeks.

## Migrating from v6 to v7

- First and foremost, create a new `lv_conf.h` based on `lv_conf_template.h`.
- To try the new version it suggested using a simulator project and see the examples.
- If you have a running project, the most difficult part of the migration is updating to the new style system. Unfortunately, there is no better way than manually updating to the new format.
- The other parts are mainly minor renames and refactoring as described above.

## 1.11 后续目标 (Roadmap)

This is a summary for planned new features and a collection of ideas. This list indicates only the current intention and it can be changed.

### 1.11.1 v8.2

See #2790

### 1.11.2 Ideas

- Reconsider color format management for run time color format setting, and custom color format usage. (Also [RGB888](#))
- Make gradients more versatile
- Image transformations matrix
- Switch to RGBA colors in styles
- Consider direct binary font format support
- Simplify `groups`. Discussion is [here](#).
- `lv_mem_alloc_aligned(size, align)`
- Text node. See #1701
- CPP binding. See [Forum](#)
- Optimize font decompression
- Need static analyze (via [coverity.io](#) or something else)



- Support dot\_begin and dot\_middle long modes for labels
- Add new label alignment modes. [#1656](#)
- Support larger images: [#1892](#)
- Curved text on path
- Variable binding improvements like Redux?
- Functional programming support, pure view? See [here](#)