100ASK LVGL Chinese Documentation 8.0

100ASK LVGL community

Jul 20, 2021

CONTENTS

CHAPTER

ONE

INTRODUCTION ????

1.1 Key features ??????

- ??????????????????????????
- ????????????????????????
- ????????????????????????????
- ??????UTF-8??
- ??????????????TFT?????????
- ??????????????css???
- ????????????????????
- ??????????????(64kb Flash, 16kb RAM)
- ??????????????GPU????????
- ????????????????????
- ?C??????????(c++??)
- ??????????????PC???????GUI??
- ???MicroPython
- ?????????GUI??????
- ????????PDF?????
- ????????????????????

1.2 Requirements??????

1.3 License?????

LVGL ?????????????? MIT license ????????? ????????????????????????

???????????????????? My projects ?????? lvgl.io ????????????????????????????????????

??????????? LVGL???

??? LVGL ????????????????? ?? ?????? ??????? ????????????????????????? LVGL ????????????????????????????

1.4 Repository layout ??????

LVGL ??????????? GitHub?https://github.com/lvgl

??????????????????

- lvgl lvgl???
- lv_examples ????????
- lv_drivers ??????????
- docs ???????? (https://docs.lvgl.io)
- blog ???????? (https://blog.lvgl.io)
- sim ??????????? (https://sim.lvgl.io)
- lv_sim_... ????? IDE ??????????
- lv_port_... LVGL ???????
- lv_binding_.. ?????????
- lv_... ????????

?? lvgl, lv_examples ? lv_drivers ????????????????

1.5 Release policy??????

????????????????

- ??? API ????????? ??? v5.0.0, v6.0.0
- ?????????????????? ??? v6.1.0, v6.2.0
- ?????????????????? ??? v6.1.1, v6.1.2

1.5.1 Branches????

????????????????

- `master` ????????????????
- `dev` ???????????????????? `master` ??
- `release/vX` ??????????

1.5.2 Release cycle??????

LVGL ??????? 2 ?????????????????????

1. ? `master` ?????????????????????????????
2. `master` ????? `release/vX`
3. ??????? `dev` ??? `master`
4. ????? 2 ??????? `master` ?????
5. Bug ???????? `master`

6. 2 ⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰

1.5.3 Tags⍰⍰⍰⍰⍰⍰

⍰⍰⍰⍰⍰⍰⍰⍰ `vX.Y.Z` ⍰⍰⍰⍰⍰⍰

1.5.4 Changelog⍰⍰⍰⍰⍰⍰

⍰⍰⍰⍰⍰ CHANGELOG.md ⍰⍰

1.5.5 Side projects

⍰⍰(docs)⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰ lvgl ⍰`(master)` ⍰⍰⍰ ⍰⍰`(latest)` ⍰⍰

1.5.6 Version support⍰⍰⍰⍰⍰⍰

⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰ `release/v6` ⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰
⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰
⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰ 1 ⍰⍰

1.6 FAQ⍰⍰⍰⍰⍰⍰

1.6.1 Where can I ask questions?⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰

⍰⍰⍰⍰⍰⍰⍰⍰https://forum.lvgl.io/⍰

⍰⍰⍰⍰ GitHub issues ⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰

1.6.2 Is my MCU/hardware supported?⍰LVGL⍰⍰⍰⍰⍰⍰ MCU/⍰⍰⍰⍰

LVGL ⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰SPI⍰RGB ⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰ *⍰⍰* ⍰ MCU⍰

⍰⍰⍰⍰

- “⍰⍰”⍰ MCU⍰⍰ STM32F⍰STM32H⍰NXP Kinetis⍰LPC⍰IMX⍰dsPIC33⍰PIC32 ⍰⍰
- ⍰⍰⍰GSM⍰WiFi ⍰⍰⍰⍰ Nordic NRF ⍰ Espressif ESP32
- Linux ⍰⍰⍰⍰⍰⍰ /dev/fb0⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰ Raspberry Pi
- ⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰ MCU ⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰

1.6.3 Is my display supported?[illegible]

LVGL [illegible] [illegible] LVGL [illegible]

[illegible]

- [illegible] 16 [illegible] 24 [illegible] TFT
- [illegible] HDMI [illegible]
- [illegible]
- [illegible]
- [illegible] LED [illegible]
- [illegible]/[illegible]

[illegible] *[illegible]* [illegible]

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

[illegible] `lv_tick_inc(x)` [illegible] `while(1)` [illegible] `lv_task_handler()` [illegible]

[illegible] *Tick* [illegible] *[illegible](Task handler)* [illegible]

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

[illegible] "*[illegible]*" [illegible] `lv_disp_flush_ready(drv)` [illegible]

1.6.6 Why I see only garbage on the screen?[illegible]

[illegible] LVGL [illegible]

```
#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 &lt; BUF_H; y++) {
    lv_color_t c = lv_color_mix(LV_COLOR_BLUE, LV_COLOR_RED, (y * 255) / BUF_H);
    for(x = 0; x &lt; BUF_W; x++){
        (*buf_p) =  c;
        buf_p++;
    }
}

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

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

?? LVGL ?????????????????????? *lv_conf.h* ?? `LV_COLOR_DEPTH` ?

???? SPI??????????????? 16 ????????? *lv_conf.h* ??? `LV_COLOR_16_SWAP  1` ????????????????

1.6.8 How to speed up my UI?????????????

- ???? MCU ????????????????????
- ??????????
- ?? 2 ?????????????? DMA????????????????
- ?????? SPI ??????????????????????????
- ?????????? SPI ???????????????????????????????
- ??????????? RAM??????? SRAM????? LVGL ????????????????????

1.6.9 How to reduce flash/ROM usage?????????/ROM??????

???? *lv_conf.h* ????????????????????????GPU ?????????

???????? GCC???????

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

????????????????????????

1.6.10 How to reduce the RAM usage??????????

?? *??????* ??? ?? *lv_conf.h* ?? `LV_MEM_SIZE` ?????????????????? ??????? `LV_MEM_SIZE` ????????????????????????????

1.6.11 How to work with an operating system?????????????

CHAPTER

TWO

GET STARTED

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

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

2.1 Quick overview

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

2.1.1 Get started in a simulator

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

LVGL is ported to many IDEs to be sure you will find your 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 make some experience with LVGL immediately.

2.1.2 Add LVGL into your project

If you rather want to 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 seed 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_darw_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_SER / 10);␣
↪ /*Initialize the display buffer.*/
```

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

```
lv_disp_drv_t disp_drv;               /*Descriptor of a display driver*/
lv_disp_drv_init(&disp_drv);          /*Basic initialization*/
disp_drv.flush_cb = my_disp_flush;    /*Set your driver function*/
disp_drv.buffer = &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.hor_res = MY_DISP_VER_RES;   /*Set the verizontal 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 a 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 touch pad:

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

(continues on next page)

(continued from previous page)

```
bool 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 Operation system task. It will redraw the screen if required, handle input devices, animation etc.

For a more detailed guide go to the Porting section.

2.1.3 Learn the basics

Widgets

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

Every object has a parent object where it is create. For example if a label is created on a button, the button is the parent of label.

The child object moves with the parent and if the parent is deleted the children will be deleted too.

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

A Screen is the "root" parent. You can have any number of screens.

To get the current screen call `lv_scr_act()`, and to load a screen use `lv_scr_load(scr1)`.

You can create a new object with `lv_<type>_create(parent)`. 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 attribute `lv_obj_set_<paramters_name>(obj, <value>)` function can be used. For example:

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

The widgets have type specific parameters too which can be set by `lv_<widget_type>_set_<paramters_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 if 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.

It should look 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");
}
```

Instead of `LV_EVENT_CLICKED` `LV_EVENT_ALL` can be used too to call the callback for any event.

From `lv_event_t * e` the current event code can be get 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 different parts. (See below)

To learn which parts are used by which object read the widgets' documentation.

States

The 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` eing pressed
- `LV_STATE_SCROLLED` Being scrolled
- `LV_STATE_DISABLED` Disabled

For example, if you press an object it will automatically goes to `LV_STATE_FOCUSED` and `LV_STATE_PRESSED` state and when you release it, the `LV_STATE_PRESSED` state will be removed.

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

To manually add remove the states use

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

Styles

Styles contains properties such as background color, border width, font, etc to describe the appearance of the objects.

The styles are `lv_style_t` variables. Only their pointer is saved in the objects so they need to be static or global. Before using a style it needs to be initialized with `lv_style_init(&style1)`. After that properties can be added. 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 go here.

The styles are assigned to 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*/
```

The styles can be cascaded (similarly to CSS). It means you can add more styles to a part of an object. For example `style_btn` can set a default button appearance, and `style_btn_red` can overwrite 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. If the property is not defined even in the default state a default value is used.

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

Local style properties also can be added to the objects. It creates a style is inside the object that 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 of the objects. The styles from the themes are applied automatically when the objects are created.

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

2.1.4 Examples

2.1.5 Micropython

Learn more about *Micropython*.

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

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

2.2 Simulator on PC

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

Simulator on the PC have the following advantages:

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

2.2.1 Select an IDE

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

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

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

2.2.2 Set-up Eclipse CDT

Install Eclipse CDT

Eclipse CDT is a C/C++ IDE.

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

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

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

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

Install SDL 2

The PC simulator uses the SDL 2 cross platform library to simulate a TFT display and a touch pad.

Linux

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

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

Windows

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

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

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

OSX

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

If something is not working, then please refer this tutorial to get started with SDL.

Pre-configured project

A pre-configured graphics library project (based on the latest release) is always available to get started easily. You can find the latest one on GitHub. (Please note that, the project is configured for Eclipse CDT).

Add the pre-configured project to Eclipse CDT

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

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

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

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

Compile and Run

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

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

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

2.3 STM32

TODO

2.4 NXP

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.

2.4.1 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.

2.4.2 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 PXP engine. In order to use CPU time while PXP is running, RTOS is required to block the LVGL drawing thread and switch to another task, or simply to idle task, where CPU could be suspended to save power.

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 ( ; ; ) ;
  }
#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. Structure `lv_nxp_pxp_cfg_t` with callback pointers is used as a parameter for `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 defined value will be processed by CPU, areas bigger than the threshold will be processed by PXP. These thresholds may be defined as a preprocessor variables. Default values are defined 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 (OPA > LV_OPA_MAX)
  - `GPU_NXP_PXP_BLIT_OPA_SIZE_LIMIT`: size threshold for image BLIT and BLIT with color keying with transparency (OPA < LV_OPA_MAX)
  - `GPU_NXP_PXP_FILL_SIZE_LIMIT`: size threshold for fill operation (OPA > LV_OPA_MAX)
  - `GPU_NXP_PXP_FILL_OPA_SIZE_LIMIT`: size threshold for fill operation with transparency (OPA < LV_OPA_MAX)

2.5 Espressif (ESP32)

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

2.5.1 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.

2.5.2 Use LVGL in your ESP32 project

Prerequisites

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

Get LVGL

You are suggested to add LVGL 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 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`.

2.5.3 Use lvgl_esp32_drivers in your project

You are suggested to 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.

2.6 Arduino

The core LVGL library and the examples are directly available as Arduino libraries.

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

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

2.6.1 Get the LVGL Ardunio library

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

2.6.2 Set up drivers

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

- `User_Setup.h`
- or by selecting a configuration in the `User_Setup_Select.h`

Both files are located in `TFT_eSPI` library's folder.

2.6.3 Configure LVGL

LVGL has its own configuration file called `lv_conf.h`. When LVGL is installed the followings needs to be done to configure it:

1. Go to directory of the installed Arduino libraries
2. Go to `lvgl` and copy `lv_conf_template.h` as `lv_conf.h` 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 resolution of your display in `LV_HOR_RES_MAX` and `LV_VER_RES_MAX`
5. Set the color depth of you display in `LV_COLOR_DEPTH`
6. Set `LV_TICK_CUSTOM 1`

2.6.4 Configure the examples

`lv_examples` can be configures similarly to LVGL but it's configuration file is called `lv_ex_conf.h`.

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

2.6.5 Initialize LVGL and run an example

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

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

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

2.6.6 Debugging and logging

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

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

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

2.7 Micropython

2.7.1 What is Micropython?

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

Highlights of Micropython

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

2.7.2 Why Micropython + LVGL?

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

Here are some advantages of using LVGL in Micropython:

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

Micropython + LVGL could be used for:

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

2.7.3 So what does it look like?

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

Let's dive right into an example!

A simple example

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

2.7.4 How can I use it?

Online Simulator

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

Click here to experiment on the online simulator

Hello World

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

PC Simulator

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

Click here to know more information about building and running the unix port

Embedded platform

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

2.7.5 Where can I find more information?

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

2.8 NuttX RTOS

2.8.1 What is NuttX?

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

Highlights of NuttX

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

2.8.2 Why NuttX + LVGL?

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

Here are some advantages of using LVGL in NuttX

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

NuttX + LVGL could be used for

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

2.8.3 How to get started with NuttX and LVGL?

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

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

Let's to use Linux and example, for Windows

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

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

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

Clone the NuttX and Apps repositories:

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

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

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

If everything went fine you should have now the file `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␣
↪halt" -c "flash write_image erase nuttx.bin 0x08000000"
```

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

```
nsh> lvgldemo
```

2.8.4 Where can I find more information?

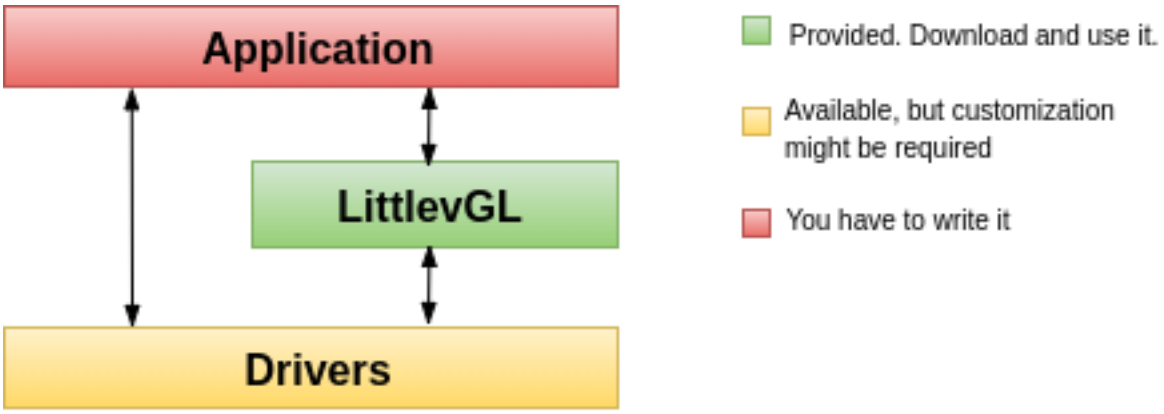
- On the LVGL on LPCXpresso54628
- NuttX mailing list Apache NuttX Mailing List

CHAPTER

THREE

PORTING

3.1 System overview



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

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

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

Depending on the MCU, there are two typical hardware set-ups. One with built-in LCD/TFT driver periphery and another without it. In both cases, a frame buffer will be required to store the current image of the screen.

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

3.2 Set-up a project

3.2.1 Get the library

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

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

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

3.2.2 Configuration file

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

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

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

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

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

3.2.3 Initialization

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

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

3.3 Display interface

To set up a display an `lv_disp_draw_buf_t` and an `lv_disp_drv_t` variables 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 drawing related things.

3.3.1 Draw buffer

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

A draw 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) */
lv_disp_draw_buf_init(&disp_buf, buf_1, buf_2, MY_DISP_HOR_RES*10);
```

Note that `lv_disp_draw_buf_t` needs to be static, global or dynamically allocated and not a local variable destroyed if goes out of the scope.

As you can see the draw buffer can be smaller than the screen. In this case, the larger areas will be redrawn in smaller parts 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 at least 1/10 screen sized.

If only **one buffer** is used LVGL draws the content of the screen into that draw buffer and sends it to the display.

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

In the display driver (`lv_disp_drv_t`) the `full_refresh` bit can be enabled to force LVGL always redraw the whole screen. It works in both *one buffer* and *two buffers* modes.

If `full_refresh` is enabled and 2 screen sized draw buffers are provided, LVGL work as "traditional" double buffering. It means in `flush_cb` only the address of the frame buffer needs to be changed to provided pointer (`color_p` parameter). This configuration should be used if the MCU has LCD controller periphery and not with an external display controller (e.g. ILI9341 or SSD1963).

You can measure the performance of different draw buffer configurations using the benchmark example.

3.3.2 Display driver

Once the buffer initialization is ready a `lv_disp_drv_t` display drivers need to be

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

Note that `lv_disp_drv_t` needs to be static, global or dynamically allocated and not a local variable destroyed if goes out of the scope.

Mandatory fields

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

- `draw_buf` pointer to an initialized `lv_disp_draw_buf_t` variable.
- `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 which is the last chunk of rendering use `lv_disp_flush_is_last(&disp_drv)`.
- `hor_res` horizontal resolution of the display in pixels.
- `ver_res` vertical resolution of the display in pixels.

Optional fields

There are some optional data fields:

- `color_chroma_key` A color which will be drawn as transparent on chrome keyed images. Set to `LV_COLOR_CHROMA_KEY` by default from `lv_conf.h`.
- `user_data` A custom `void` user data for the driver..
- `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` needs to enabled in `lv_conf.h` and requires `LV_COLOR_DEPTH 32`.

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

- `rounder_cb` Round the coordinates of areas to redraw. E.g. a 2x2 px can be converted to 2x8. It can be used if the display controller can refresh only areas with specific height or width (usually 8 px height with monochrome displays).
- `set_px_cb` a custom function to write the draw buffer. It can be used to store the pixels more compactly i nthe 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. Rendering with `set_px_cb` is slower than normal rendering.
- `monitor_cb` A callback function that tells how many pixels were refreshed in how much time.
- `clean_dcache_cb` A callback for cleaning any caches related to the display.

To use a GPU the following callbacks can be used:

- `gpu_fill_cb` fill an area in the memory with a color.
- `gpu_wait_cb` if any GPU function return, while the GPU is still working, LVGL will use this function when required the be 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)
{
    /*The most simple case (but also the slowest) to put all pixels to the screen one-
↪by-one*/
    int32_t x, y;
    for(y = area->y1; y <= area->y2; y++) {
        for(x = area->x1; x <= area->x2; x++) {
            put_px(x, y, *color_p)
            color_p++;
        }
    }

    /* IMPORTANT!!!
     * Inform the graphics library that you are ready with the flushing*/
    lv_disp_flush_ready(disp_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*/

    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 make the area to start only on 8th rows and have Nx8 pixel height:*/
   area->y1 = area->y1 & 0x07;
```

(continues on next page)

(continued from previous page)

```
   area->y2 = (area->y2 & 0x07) + 8;
}

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

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

void my_clean_dcache_cb(lv_disp_drv_t * disp_drv, uint32)
{
  /* Example for Cortex-M (CMSIS) */
  SCB_CleanInvalidateDCache();
}
```

3.3.3 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, which is why hardware rotation is also available. In this mode, LVGL draws into the buffer as though your screen now has the width and height inverted. 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(disp, 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.

3.3.4 Further reading

See lv_port_disp_template.c for a template for your own driver.

Check out the *Drawing* section to learn more about how rendering works in LVGL.

3.3.5 API

@description Display Driver HAL interface header file

Typedefs

typedef struct *_lv_disp_drv_t* **lv_disp_drv_t**

Display Driver structure to be registered by HAL. Only its pointer will be saved in `lv_disp_t` so it should be declared as `static lv_disp_drv_t my_drv` or allocated dynamically.

typedef struct *_lv_disp_t* **lv_disp_t**

Display structure.

Note: `lv_disp_drv_t` should be the first member of the structure.

Enums

enum **lv_disp_rot_t**

Values:

enumerator **LV_DISP_ROT_NONE**

enumerator **LV_DISP_ROT_90**

enumerator **LV_DISP_ROT_180**

enumerator **LV_DISP_ROT_270**

Functions

void **lv_disp_drv_init**(*lv_disp_drv_t* *driver)

Initialize a display driver with default values. It is used to have known values in the fields and not junk in memory. After it you can safely set only the fields you need.

Parameters **driver** -- pointer to driver variable to initialize

void **lv_disp_draw_buf_init**(*lv_disp_draw_buf_t* *draw_buf, void *buf1, void *buf2, uint32_t size_in_px_cnt)

Initialize a display buffer

Parameters

- **draw_buf** -- pointer *lv_disp_draw_buf_t* variable to initialize

- **buf1** -- A buffer to be used by LVGL to draw the image. Always has to specified and can't be NULL. Can be an array allocated by the user. E.g. `static lv_color_t disp_buf1[1024 * 10]` Or a memory address e.g. in external SRAM
- **buf2** -- Optionally specify a second buffer to make image rendering and image flushing (sending to the display) parallel. In the `disp_drv->flush` you should use DMA or similar hardware to send the image to the display in the background. It lets LVGL to render next frame into the other buffer while previous is being sent. Set to `NULL` if unused.
- **size_in_px_cnt** -- size of the `buf1` and `buf2` in pixel count.

lv_disp_t ***lv_disp_drv_register**(*lv_disp_drv_t* *driver)

Register an initialized display driver. Automatically set the first display as active.

Parameters **driver** -- pointer to an initialized 'lv_disp_drv_t' variable. Only its pointer is saved!

Returns pointer to the new display or NULL on error

void **lv_disp_drv_update**(*lv_disp_t* *disp, *lv_disp_drv_t* *new_drv)

Update the driver in run time.

Parameters

- **disp** -- pointer to a display. (return value of `lv_disp_drv_register`)
- **new_drv** -- pointer to the new driver

void **lv_disp_remove**(*lv_disp_t* *disp)

Remove a display

Parameters **disp** -- pointer to display

void **lv_disp_set_default**(*lv_disp_t* *disp)

Set a default display. The new screens will be created on it by default.

Parameters **disp** -- pointer to a display

lv_disp_t ***lv_disp_get_default**(void)

Get the default display

Returns pointer to the default display

lv_coord_t **lv_disp_get_hor_res**(*lv_disp_t* *disp)

Get the horizontal resolution of a display

Parameters **disp** -- pointer to a display (NULL to use the default display)

Returns the horizontal resolution of the display

lv_coord_t **lv_disp_get_ver_res**(*lv_disp_t* *disp)

Get the vertical resolution of a display

Parameters **disp** -- pointer to a display (NULL to use the default display)

Returns the vertical resolution of the display

bool **lv_disp_get_antialiasing**(*lv_disp_t* *disp)

Get if anti-aliasing is enabled for a display or not

Parameters **disp** -- pointer to a display (NULL to use the default display)

Returns true: anti-aliasing is enabled; false: disabled

lv_coord_t **lv_disp_get_dpi**(const *lv_disp_t* *disp)

Get the DPI of the display

Parameters **disp** -- pointer to a display (NULL to use the default display)

Returns dpi of the display

void **lv_disp_set_rotation**(*lv_disp_t* *disp, *lv_disp_rot_t* rotation)

Set the rotation of this display.

Parameters

- **disp** -- pointer to a display (NULL to use the default display)
- **rotation** -- rotation angle

lv_disp_rot_t **lv_disp_get_rotation**(*lv_disp_t* *disp)

Get the current rotation of this display.

Parameters **disp** -- pointer to a display (NULL to use the default display)

Returns rotation angle

lv_disp_t ***lv_disp_get_next**(*lv_disp_t* *disp)

Get the next display.

Parameters **disp** -- pointer to the current display. NULL to initialize.

Returns the next display or NULL if no more. Give the first display when the parameter is NULL

lv_disp_draw_buf_t ***lv_disp_get_draw_buf**(*lv_disp_t* *disp)

Get the internal buffer of a display

Parameters **disp** -- pointer to a display

Returns pointer to the internal buffers

struct **lv_disp_draw_buf_t**

#include <lv_hal_disp.h> Structure for holding display buffer information.

Public Members

void ***buf1**

First display buffer.

void ***buf2**

Second display buffer.

void ***buf_act**

uint32_t **size**

lv_area_t **area**

int **flushing**

int **flushing_last**

uint32_t **last_area**

uint32_t **last_part**

struct **_lv_disp_drv_t**

#include <lv_hal_disp.h> Display Driver structure to be registered by HAL. Only its pointer will be saved in `lv_disp_t` so it should be declared as `static lv_disp_drv_t my_drv` or allocated dynamically.

Public Members

lv_coord_t **hor_res**
Horizontal resolution.

lv_coord_t **ver_res**
Vertical resolution.

lv_disp_draw_buf_t ***draw_buf**
Pointer to a buffer initialized with `lv_disp_draw_buf_init()`. LVGL will use this buffer(s) to draw the screens contents

uint32_t **full_refresh**
1: Always make the whole screen redrawn

uint32_t **sw_rotate**
1: use software rotation (slower)

uint32_t **antialiasing**
1: anti-aliasing is enabled on this display.

uint32_t **rotated**
1: turn the display by 90 degree.

> **Warning:** Does not update coordinates for you!

uint32_t **screen_transp**

uint32_t **dpi**
Handle if the screen doesn't have a solid (opa == LV_OPA_COVER) background. Use only if required because it's slower.

void (***flush_cb**)(struct *_lv_disp_drv_t* *disp_drv, const lv_area_t *area, lv_color_t *color_p)
DPI (dot per inch) of the display. Default value is `LV_DPI_DEF`. MANDATORY: Write the internal buffer (draw_buf) to the display. 'lv_disp_flush_ready()' has to be called when finished

void (***rounder_cb**)(struct *_lv_disp_drv_t* *disp_drv, lv_area_t *area)
OPTIONAL: Extend the invalidated areas to match with the display drivers requirements E.g. round `y` to, 8, 16 ..) on a monochrome display

void (***set_px_cb**)(struct *_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)
OPTIONAL: Set a pixel in a buffer according to the special requirements of the display Can be used for color format not supported in LittelvGL. E.g. 2 bit -> 4 gray scales

> **Note:** Much slower then drawing with supported color formats.

void (***monitor_cb**)(struct *_lv_disp_drv_t* *disp_drv, uint32_t time, uint32_t px)
OPTIONAL: Called after every refresh cycle to tell the rendering and flushing time + the number of flushed

pixels

void (*`wait_cb`)(struct *_lv_disp_drv_t* *disp_drv)
OPTIONAL: Called periodically while lvgl waits for operation to be completed. For example flushing or GPU User can execute very simple tasks here or yield the task

void (*`clean_dcache_cb`)(struct *_lv_disp_drv_t* *disp_drv)
OPTIONAL: Called when lvgl needs any CPU cache that affects rendering to be cleaned

void (*`gpu_wait_cb`)(struct *_lv_disp_drv_t* *disp_drv)
OPTIONAL: called to wait while the gpu is working

void (*`drv_update_cb`)(struct *_lv_disp_drv_t* *disp_drv)
OPTIONAL: called when driver parameters are updated

void (*`gpu_fill_cb`)(struct *_lv_disp_drv_t* *disp_drv, lv_color_t *dest_buf, lv_coord_t dest_width, const lv_area_t *fill_area, lv_color_t color)
OPTIONAL: Fill a memory with a color (GPU only)

lv_color_t `color_chroma_key`
On CHROMA_KEYED images this color will be transparent. `LV_COLOR_CHROMA_KEY` by default. (lv_conf.h)

void *`user_data`
Custom display driver user data

struct `_lv_disp_t`
#include <lv_hal_disp.h> Display structure.

Note: `lv_disp_drv_t` should be the first member of the structure.

Public Members

lv_disp_drv_t *`driver`
< Driver to the display A timer which periodically checks the dirty areas and refreshes them

lv_timer_t *`refr_timer`
The theme assigned to the screen

struct *_lv_theme_t* *`theme`

struct *_lv_obj_t* **`screens`
Screens of the display Array of screen objects.

struct *_lv_obj_t* *`act_scr`
Currently active screen on this display

struct *_lv_obj_t* *`prev_scr`
Previous screen. Used during screen animations

struct *_lv_obj_t* ***scr_to_load***
The screen prepared to load in lv_scr_load_anim

struct *_lv_obj_t* ***top_layer***

See *lv_disp_get_layer_top*

struct *_lv_obj_t* ***sys_layer***

See *lv_disp_get_layer_sys*

uint32_t ***screen_cnt***

uint8_t ***del_prev***
1: Automatically delete the previous screen when the screen load animation is ready

lv_opa_t ***bg_opa***
Opacity of the background color or wallpaper

lv_color_t ***bg_color***
Default display color when screens are transparent

const void ****bg_img***
An image source to display as wallpaper

lv_area_t ***inv_areas***[LV_INV_BUF_SIZE]
Invalidated (marked to redraw) areas

uint8_t ***inv_area_joined***[LV_INV_BUF_SIZE]

uint16_t ***inv_p***

uint32_t ***last_activity_time***
Last time when there was activity on this display

3.4 Input device interface

3.4.1 Types of input devices

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

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

`type` can be

- `LV_INDEV_TYPE_POINTER` touchpad or mouse
- `LV_INDEV_TYPE_KEYPAD` keyboard or keypad
- `LV_INDEV_TYPE_ENCODER` encoder with left/right 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 of 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 4 things:

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

In short, the Encoder input devices work like this:

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

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

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

...

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 encoder click with period specified in `indev_drv.long_press_rep_time`.

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

...

bool encoder_with_keys_read(lv_indev_drv_t * drv, lv_indev_data_t*data){
```

(continues on next page)

(continued from previous page)

```
  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();
  }

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

Button

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

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

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

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

...

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*/
}
```

3.4.2 Other features

Parameters

The default value of the following parameters can 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_rimer` 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 from its type). It allows making 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 lastly created or the explicitly selected (using `lv_disp_set_default()`) display. The associated display is stored and can be changed in `disp` field of the driver.

Event driven reading

By default LVGL calls `read_cb` periodically. This way there is a chance that some user gestures are missed.

To solve this you write an event driven driver for your input device that buffers measured data. In `read_cb` you can set the buffered data instead of reading the input device. You can set the `data->continue_reding` flag to LVGL there is more data to read and call `read_cb` again.

3.4.3 API

@description Input Device HAL interface layer header file

Typedefs

typedef struct *_lv_indev_drv_t* **`lv_indev_drv_t`**
Initialized by the user and registered by 'lv_indev_add()'

typedef struct *_lv_indev_proc_t* **`lv_indev_proc_t`**
Run time data of input devices Internally used by the library, you should not need to touch it.

typedef struct *_lv_indev_t* **`lv_indev_t`**
The main input device descriptor with driver, runtime data ('proc') and some additional information

Enums

enum **lv_indev_type_t**
Possible input device types

Values:

enumerator **LV_INDEV_TYPE_NONE**
Uninitialized state

enumerator **LV_INDEV_TYPE_POINTER**
Touch pad, mouse, external button

enumerator **LV_INDEV_TYPE_KEYPAD**
Keypad or keyboard

enumerator **LV_INDEV_TYPE_BUTTON**
External (hardware button) which is assigned to a specific point of the screen

enumerator **LV_INDEV_TYPE_ENCODER**
Encoder with only Left, Right turn and a Button

enum **lv_indev_state_t**
States for input devices

Values:

enumerator **LV_INDEV_STATE_RELEASED**

enumerator **LV_INDEV_STATE_PRESSED**

Functions

void **lv_indev_drv_init**(*lv_indev_drv_t* *driver)
Initialize an input device driver with default values. It is used to surly have known values in the fields ant not memory junk. After it you can set the fields.

Parameters **driver** -- pointer to driver variable to initialize

lv_indev_t ***lv_indev_drv_register**(*lv_indev_drv_t* *driver)
Register an initialized input device driver.

Parameters **driver** -- pointer to an initialized 'lv_indev_drv_t' variable (can be local variable)

Returns pointer to the new input device or NULL on error

void **lv_indev_drv_update**(*lv_indev_t* *indev, *lv_indev_drv_t* *new_drv)
Update the driver in run time.

Parameters

- **indev** -- pointer to a input device. (return value of `lv_indev_drv_register`)
- **new_drv** -- pointer to the new driver

lv_indev_t ***lv_indev_get_next**(*lv_indev_t* *indev)
Get the next input device.

Parameters **indev** -- pointer to the current input device. NULL to initialize.

Returns the next input devise or NULL if no more. Give the first input device when the parameter is NULL

void **_lv_indev_read**(*lv_indev_t* *indev, *lv_indev_data_t* *data)

Read data from an input device.

Parameters

- **indev** -- pointer to an input device
- **data** -- input device will write its data here

struct **lv_indev_data_t**

#include <lv_hal_indev.h> Data structure passed to an input driver to fill

Public Members

lv_point_t **point**

For LV_INDEV_TYPE_POINTER the currently pressed point

uint32_t **key**

For LV_INDEV_TYPE_KEYPAD the currently pressed key

uint32_t **btn_id**

For LV_INDEV_TYPE_BUTTON the currently pressed button

int16_t **enc_diff**

For LV_INDEV_TYPE_ENCODER number of steps since the previous read

lv_indev_state_t **state**

LV_INDEV_STATE_REL or LV_INDEV_STATE_PR

bool **continue_reading**

Call the read callback until it's set to true

struct **_lv_indev_drv_t**

#include <lv_hal_indev.h> Initialized by the user and registered by 'lv_indev_add()'

Public Members

lv_indev_type_t **type**

< Input device type Function pointer to read input device data.

void (***read_cb**)(struct *_lv_indev_drv_t* *indev_drv, *lv_indev_data_t* *data)

void (***feedback_cb**)(struct *_lv_indev_drv_t**, uint8_t)

Called when an action happened on the input device. The second parameter is the event from `lv_event_t`

void ***user_data**

struct *_lv_disp_t* ***disp**
< Pointer to the assigned display Timer to periodically read the input device

lv_timer_t ***read_timer**
Number of pixels to slide before actually drag the object

uint8_t **scroll_limit**
Drag throw slow-down in [%]. Greater value means faster slow-down

uint8_t **scroll_throw**
At least this difference should between two points to evaluate as gesture

uint8_t **gesture_min_velocity**
At least this difference should be to send a gesture

uint8_t **gesture_limit**
Long press time in milliseconds

uint16_t **long_press_time**
Repeated trigger period in long press [ms]

uint16_t **long_press_repeat_time**

struct **_lv_indev_proc_t**
#include <lv_hal_indev.h> Run time data of input devices Internally used by the library, you should not need to touch it.

Public Members

lv_indev_state_t **state**
Current state of the input device.

uint8_t **long_pr_sent**

uint8_t **reset_query**

uint8_t **disabled**

uint8_t **wait_until_release**

lv_point_t **act_point**
Current point of input device.

lv_point_t **last_point**
Last point of input device.

lv_point_t **last_raw_point**
Last point read from read_cb.

lv_point_t **vect**
Difference between `act_point` and `last_point`.

lv_point_t **scroll_sum**

lv_point_t **scroll_throw_vect**

lv_point_t **scroll_throw_vect_ori**

struct *_lv_obj_t* ***act_obj**

struct *_lv_obj_t* ***last_obj**

struct *_lv_obj_t* ***scroll_obj**

struct *_lv_obj_t* ***last_pressed**

lv_area_t **scroll_area**

lv_point_t **gesture_sum**

lv_dir_t **scroll_dir**

lv_dir_t **gesture_dir**

uint8_t **gesture_sent**

struct *_lv_indev_proc_t*::[anonymous]::[anonymous] **pointer**

lv_indev_state_t **last_state**

uint32_t **last_key**

struct *_lv_indev_proc_t*::[anonymous]::[anonymous] **keypad**

union *_lv_indev_proc_t*::[anonymous] **types**

uint32_t **pr_timestamp**
Pressed time stamp

uint32_t **longpr_rep_timestamp**
Long press repeat time stamp

struct **_lv_indev_t**
#include <lv_hal_indev.h> The main input device descriptor with driver, runtime data ('proc') and some additional information

Public Members

lv_indev_drv_t ***driver**

lv_indev_proc_t **proc**

struct *_lv_obj_t* ***cursor**
Cursor for LV_INPUT_TYPE_POINTER

struct *_lv_group_t* ***group**
Keypad destination group

const lv_point_t ***btn_points**
Array points assigned to the button ()screen will be pressed here by the buttons

3.5 Tick interface

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

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

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

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

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

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

3.5.1 API

Provide access to the system tick with 1 millisecond resolution

Functions

uint32_t **lv_tick_get**(void)
Get the elapsed milliseconds since start up

Returns the elapsed milliseconds

uint32_t **lv_tick_elaps**(uint32_t prev_tick)
Get the elapsed milliseconds since a previous time stamp

Parameters **prev_tick** -- a previous time stamp (return value of *lv_tick_get()*)

Returns the elapsed milliseconds since 'prev_tick'

3.6 Task Handler

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

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

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

Example:

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

To learn more about task visit the Tasks section.

3.7 Sleep management

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

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

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

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

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

3.8 Operating system and interrupts

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

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

- In *events*. Learn more in *Events*.
- In *lv_tasks*. Learn more in Tasks.

3.8.1 Tasks and threads

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

3.8.2 Interrupts

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

3.9 Logging

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

3.9.1 Log level

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

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

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

3.9.2 Logging with printf

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

3.9.3 Custom log function

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

For example:

```
void my_log_cb(lv_log_level_t level, const char * file, uint32_t line, const char *␣
↪fn_name, const char * dsc)
{
  /*Send the logs via serial port*/
  if(level == LV_LOG_LEVEL_ERROR) serial_send("ERROR: ");
  if(level == LV_LOG_LEVEL_WARN)  serial_send("WARNING: ");
```

(continues on next page)

(continued from previous page)

```
  if(level == LV_LOG_LEVEL_INFO)  serial_send("INFO: ");
  if(level == LV_LOG_LEVEL_TRACE) serial_send("TRACE: ");

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

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

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

...


lv_log_register_print_cb(my_log_cb);
```

3.9.4 Add logs

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

CHAPTER

FOUR

OVERVIEW

4.1 Objects

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

Check all the *Object types* here.

All objects are referenced using an `lv_obj_t` pointer as a handle. This pointer can later be used to set or get the attributes of the object.

4.1.1 Attributes

Basic attributes

All object types share some basic attributes:

- Position
- Size
- Parent
- Styles
- Event handlers
- Etc

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

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

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

Specific attributes

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

- Minimum and maximum values
- Current value

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

```
/*Set slider specific attributes*/
lv_slider_set_range(slider1, 0, 100);                                    /*Set␣
↪the min. and max. values*/
lv_slider_set_value(slider1, 40, LV_ANIM_ON);                /*Set the current value␣
↪(position)*/
```

The API of the widgets is described in their *Documentation* but you can also check the respective header files (e.g. *widgets/lv_slider.h*)

4.1.2 Working mechanisms

Parent-child structure

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

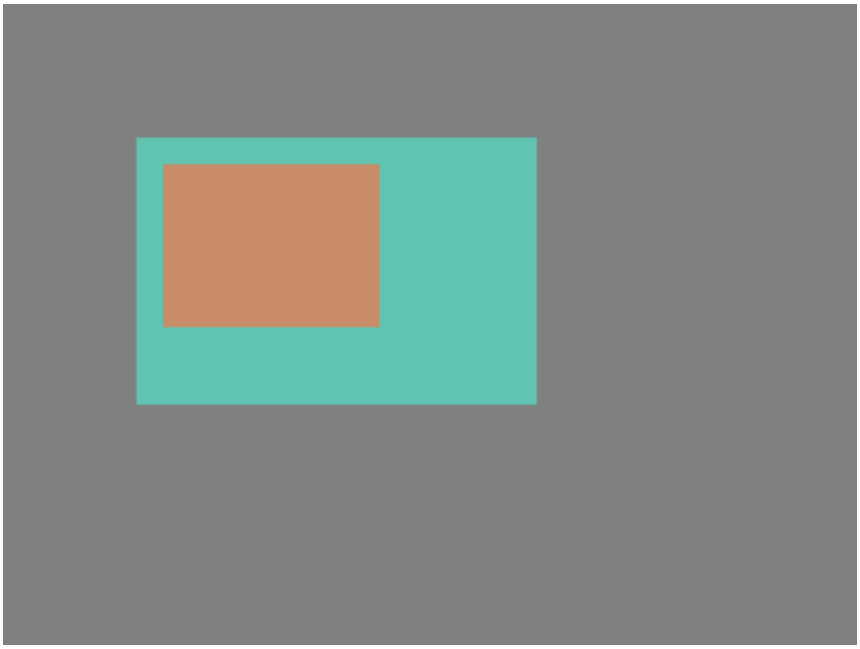
Moving together

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

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

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

Modify the position of the parent:



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

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

Visibility only on the parent

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

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

Create and delete objects

In LVGL objects can be created and deleted dynamically in run time. It means only the currently created (existing) objects consume RAM.

It allows to create a screen just when a button is clicked to open it. A delete the screen when a new screen is loaded.

Or the UI can be created based on the current environment of the device. For example create meter, charts, bars, slider etc according to the currently attached sensors.

Every widget has its own **create** function with a prototype like this:

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

In most of the cases the create functions have only a *parent* parameter that tells on which object create the new widget.

The return value is a pointer to the created object with `lv_obj_t *` type.

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

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

`lv_obj_del` will delete the object immediately. If for any reason you can't delete the object immediately you can use `lv_obj_del_async(obj)` that will perefome the deletion on hte next call of `lv_timer_handler()`. It is useful e.g. if you want to delete the parent of an object in the child's `LV_EVENT_DELETE` signal.

You can remove all the children of an object (but not the object itself) using `lv_obj_clean(obj)`.

4.1.3 Screens

Create screens

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

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

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

Get the active screen

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

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

Load screens

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

Layers

There are two automatically generated layers:

- top layer
- system layer

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

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

Read the *Layer overview* section to learn more about layers.

Load screen with animation

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

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

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

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

Handling multiple displays

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

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

Visit *Multi-display support* to learn more.

4.1.4 Parts

The widgets are built from multiple parts. For example a *Base object* uses the main and scroll bar parts but a *Slider* uses the main, the indicator and the knob parts. Parts are similar to *pseudo elements* in CSS.

The following predefined parts exist in LVGL:

- `LV_PART_MAIN` A background like rectangle*/
- `LV_PART_SCROLLBAR` The scrollbar(s)
- `LV_PART_INDICATOR` Indicator, e.g. for slider, bar, switch, or the tick box of the checkbox
- `LV_PART_KNOB` Like a handle to grab to adjust the value*/
- `LV_PART_SELECTED` Indicate the currently selected option or section
- `LV_PART_ITEMS` Used if the widget has multiple similar elements (e.g. tabel cells)*/
- `LV_PART_TICKS` Ticks on scales e.g. for a chart or meter
- `LV_PART_CURSOR` Mark a specific place e.g. text area's or chart's cursor
- `LV_PART_CUSTOM_FIRST` Custom parts can be added from here.

The main purpose of parts to allow styling the "components" of the widgets. Therefore the parts are described in more detail in the *Style overview* section.

4.1.5 States

The object can be in a combinations 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 state
- `LV_STATE_USER_1` Custom state
- `LV_STATE_USER_2` Custom state

- `LV_STATE_USER_3` Custom state
- `LV_STATE_USER_4` Custom state

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

To learn more about the states read the related section of the *Style overview*.

4.2 Positions, sizes, and layouts

4.2.1 Overview

Similarly to many other parts of LVGL, the concept of setting the coordinates were inspired by CSS. It doesn't mean a perfect copy of the standard but subsets of CSS were implemented (sometimes with minor adjustments). It shorts it means:

- the set coordinates (size, position, layouts, etc) are stored in styles
- support min-width, max-width, min-height, max-height
- have pixel, percentage, and "content" units
- x=0; y=0 coordinate means the to top-left corner of the parent plus the left/top padding plus border width
- width/height means the full size, the "content area" is smaller with padding and border width
- a subset of flexbox and grid layouts are supported

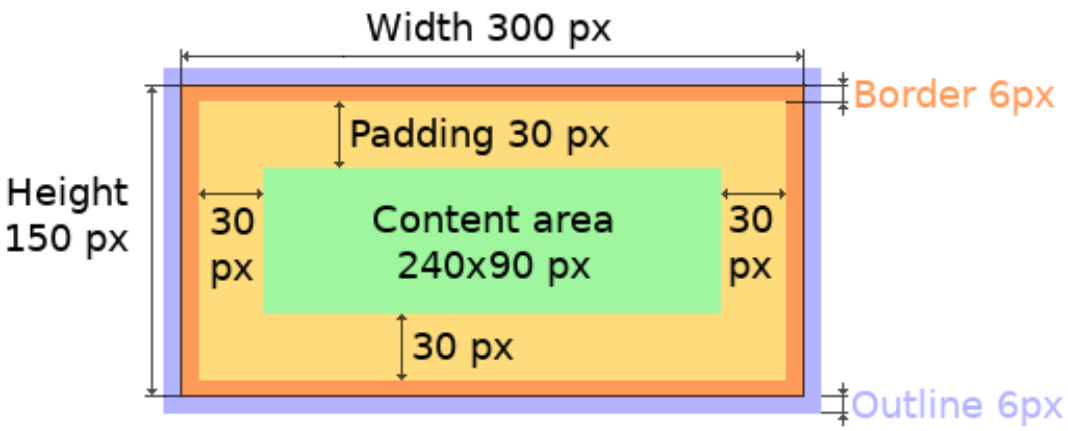
Units

- pixel: Simply a position in pixels. A simple integer always mean pixel. E.g. `lv_obj_set_x(btn, 10)`
- percentage: The percentage of the size of the object or its parent (depending on the property). The `lv_pct(value)` converts a value to percentage. E.g. `lv_obj_set_width(btn, lv_pct(50))`
- `LV_SIZE_CONTENT`: Special value to set the width/height of an object to involve all the children. Its similar to `auto` in CSS. E.g. `lv_obj_set_width(btn, LV_SIZE_CONTENT)`.

Boxing model

LVGL follows CSS's border-box model. An object's "box" is built from the following parts:

- bounding box: the width/height of the elements.
- border width: the width of the border.
- padding: space between the sides of the object and its children.
- content: the content area which size if the bounding box reduced by the border width and the size of the paddings.



The border is drawn inside the bounding box. Inside the border LVGL keeps "padding size" to place the children.

The outline is drawn outside of the bounding box.

Important notes

This section describes special cases in which LVGL's behavior might look unexpected.

Postponed coordinate calculation

LVGL doesn't recalculate all the coordinate changes immediately to improve performance. Instead, the objects are marked as "dirty" and before redrawing the screen LVGL checks if there are any "dirty" objects. If so it refreshes their position, size and layout.

In other words, if you need to get the any coordinate of an object and it the coordinates were just changed LVGL's needs to be forced to recalculate to coordinates. To do this call `lv_obj_update_layout(obj)` the size and position might depends on the parent or layout `lv_obj_update_layout` recalculates the coordinates of all objects on the screen of `obj`.

Removing styles

As it's described in the *Using styles* section the coordinates can be set via style properties too. To be more precise under the hood every style coordinate related property is stored as style a property. If you use `lv_obj_set_x(obj, 20)` LVGL saves `x=20` in the local style of the object.

It's an internal mechanism and doesn't matter much as you use LVGL. However, there is one case in which you need to aware of that. If the style(s) of an object are removed by

```
lv_obj_remove_style_all(obj)
```

or

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

The earlier set coordinates will be removed as well.

For example:

```
/*The size of obj1 will be set back to the default in the end*/
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 will have 200;100 size in the end */
lv_obj_remove_style_all(obj2);
lv_obj_set_size(obj2, 200, 100);
```

4.2.2 Position

Simple way

To simple set the x and y coordinates of an object use

```
lv_obj_set_x(obj, 10);
lv_obj_set_y(obj, 20);
lv_obj_set_pos(obj, 10, 20);         //Or in one function
```

By default the the x and y coordinates are measured from the top left corner of the parent's content area. For example if the parent has 5 pixel padding on every side, the above code will place `obj` to (15, 25) because the content area starts after the padding.

If percentage values are calculated from the parents content area size.

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

Align

In some cases it's convenient to change the origin of the positioning from the the default top left. If the origin is changed e.g. to bottom-right, the (0,0) position means: align to the bottom-right corner. To change the origin use:

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

To change the alignment and set new coordinates:

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

The following alignment options can be used:

- `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`

It quite common to align a children to the center of its parent, there fore is a dedicated function for it:

```
lv_obj_center(obj);

//Has the same effect
lv_obj_align(obj, LV_ALIGN_CENTER, 0, 0);
```

If the parent's size changes the set alignment and position of the children is applied again automatically.

The functions introduced above aligns the object to its parent. However it's also possible to align an object to an arbitrary object.

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

Besides the alignments options above the following can be used to align the object outside of the reference object:

- `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`

For example to align a label above a button and center the label is horizontally:

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

Not that - unlike with `lv_obj_align()` - `lv_obj_align_to()` can not realign the object if its coordinates or the reference object's coordinates changes.

4.2.3 Size

Simple way

The width and the height of an object can be set easily as well:

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

Percentage values are calculated based on the parent's content area size. For example to set the object's height to the screen height:

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

Size setting supports a value: `LV_SIZE_CONTENT`. It means the object's size in the respective direction will be set to involve its the children. Note that only children on the right and bottom will be considered and children on the top and left remains cropped. This limitation makes the behavior more predictable.

Object with `LV_OBJ_FLAG_HIDDEN` or `LV_OBJ_FLAG_FLOATING` will be ignored by `LV_SIZE_CONTENT` calculation.

The above functions set the size of the bounding box of the object but the size of the content area can be set as well. It means the object's bounding box will be larger with the paddings than the set size.

```
lv_obj_set_content_width(obj, 50); //The actual width: padding left + 50 + padding␣
↪right
lv_obj_set_content_height(obj, 30); //The actual width: padding top + 30 + padding␣
↪bottom
```

The size of the bounding box and the content area can be get with the following functions:

```
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);
```

4.2.4 Using styles

Under the hood the position, size and alignment properties are style properties. The above described "simple functions" hide the style related code for the sake of simplicity and set the position, size, and alignment properties in the local styles of the obejct.

However, using styles as to set the coordinates has some great advantages:

- It makes easy to set the width/height/etc for several object together with ease. E.g. all make all the sliders 100x10 pixels sized.
- It also makes possible to modify the values in one place.
- The values can be overwritten by other styles. For example `style_btn` makes the object `100x50` by default but adding `style_full_width` overwrites only the width of the object.
- The object can have different position or size in different state. E.g. 100 px wide in `LV_STATE_DEFAULT` but 120 px in `LV_STATE_PRESSED`.
- Style transitions can be used to make the coordinate changes smooth.

Here are some examples to set an object's size using a style:

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

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

As you will see below there are some other great features of size and position setting. However, to keep the LVGL's API lean only the most common coordinate setting features have a "simple" version and the more complex features can be used via styles.

4.2.5 Translation

Let's say the there are 3 buttons next to each other. Their position is set as described above. Now you want to move a buttons up a little when it's pressed.

One way to achieve this is setting a new Y coordinate for pressed state:

```
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);
```

It works but it's not really flexible because the pressed coordinate is hard-coded. If the buttons are not at y=100 `style_pressed` won't work as expected. To solve this translations can be used:

```
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);

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);
```

Translation is applied from the current position of the object.

Percentage values can be used in translations as well. The percentage is relative to the size of the object (and not to the size of the parent). For example `lv_pct(50)` will move the object with half of its width/height.

The translations is applied after the layouts are calculated. Therefore, even the layouted objects' position can be translated.

The translation actually moves the object. It means it makes the scrollbars and `LV_SIZE_CONTENT` sized objects react on the position change.

4.2.6 Transformation

Similarly to the position the size can be changed relative to the current size as well. The transformed width and height is added on both sides of the object. That is 10 px transformed width makes the object 2x10 pixel wider.

Unlike position translation, the size transformation doesn't make the object "really" larger. In other words scrollbars, layouts, `LV_SIZE_CONTENT` will not consider the transformed size. Hence size transformation if "only" a visual effect.

This code makes the a button larger when it's pressed:

```
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);
```

Min and Max size

Similarly to CSS, LVGL also support `min-width`, `max-width`, `min-height` and `max-height`. These are limits preventing an object's size to be smaller/larger then these values. They are especially useful if the size is set by percentage or `LV_SIZE_CONTENT`.

```
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
```

Percentage values can be used as well which are relative to the size of the parent's content area size.

```
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
```

4.2.7 Layout

Overview

Layouts can update the position and size of an object's children. They can be used to automatically arrange the children into a line or column, or in much more complicated forms.

The position and size set by the layout overwrites the "normal" x, y, width, and height settings.

There is only one function that is the same for every layout: `lv_obj_set_layout(obj, <LAYOUT_NAME>)` sets the layout on an object.For the further settings of the parent and children see the documentations of the given layout.

Built-in layout

LVGL comes with two very powerful layouts:

- Flexbox
- Grid

Both are heavily inspired by the CSS layouts with the same name.

Flags

There are some flags that can be used on object to affect how they behave with layouts:

- `LV_OBJ_FLAG_HIDDEN` Hidden object are ignored from layout calculations.
- `LV_OBJ_FLAG_IGNORE_LAYOUT` The object is simply ignored by the layouts. Its coordinates can be set as usual.
- `LV_OBJ_FLAG_FLOATING` Same as `LV_OBJ_FLAG_IGNORE_LAYOUT` but the object with `LV_OBJ_FLAG_FLOATING` will be ignored from `LV_SIZE_CONTENT` calculations.

These flags can be added/removed with `lv_obj_add/clear_flag(obj, FLAG);`

Adding new layouts

LVGL can be freely extended by a custom layouts like this:

```
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 required to reposition/resize the children␣
↪of "obj" */
}
```

Custom style properties can be added too that can be get and used in the update callback. For example:

```
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);
}
```

4.2.8 Examples

4.3 Styles

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

- A style is an `lv_style_t` variable which can hold properties, for example border width, text color and so on. It's similar to a `class` in CSS.
- Styles can be assigned to objects to change their appearance. During the assignment the target part (*pseudo element* in CSS) and target state (*pseudo class*) can be specified. For example add `style_blue` to the knob of a slider when it's in pressed state.
- The same style can be used by any number of objects.
- Styles can be cascaded which means multiple styles can be assigned to an object and each style can have different properties. Therefore not all properties have to be specified in style. LVLG will look for a property until a style defines it or use a default if it's not spefied by any of the styles. For example `style_btn` can result in a default gray button and `style_btn_red` can add only a `background-color=red` to overwrite the background color.
- Later added styles have higher precedence. It means if a property is specified in two styles the later added will be used.
- Some properties (e.g. text color) can be inherited from the parent(s) if it's not specified in the object.
- Objects can have local styles that have higher precedence than "normal" styles.
- Unlike CSS (where pseudo-classes describe different states, e.g. `:focus`), in LVGL a property is assigned to a given state.
- Transitions can be applied when the object changes state.

4.3.1 States

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

- `LV_STATE_DEFAULT` (0x0000) Normal, released state
- `LV_STATE_CHECKED` (0x0001) Toggled or checked state
- `LV_STATE_FOCUSED` (0x0002) Focused via keypad or encoder or clicked via touchpad/mouse
- `LV_STATE_FOCUS_KEY` (0x0004) Focused via keypad or encoder but not via touchpad/mouse
- `LV_STATE_EDITED` (0x0008) Edit by an encoder
- `LV_STATE_HOVERED` (0x0010) Hovered by mouse (not supported now)
- `LV_STATE_PRESSED` (0x0020) Being pressed
- `LV_STATE_SCROLLED` (0x0040) Being scrolled
- `LV_STATE_DISABLED` (0x0080) Disabled state
- `LV_STATE_USER_1` (0x1000) Custom state
- `LV_STATE_USER_2` (0x2000) Custom state
- `LV_STATE_USER_3` (0x4000) Custom state
- `LV_STATE_USER_4` (0x8000) Custom state

The combination states the object can be focused and pressed at the same time. It represented as `LV_STATE_FOCUSED | LV_STATE_PRESSED`.

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

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

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

1. By the default the object is in default state, so it's a simple case: the property is perfectly defined in the object's current state as white.
2. When the object is pressed there are 2 related properties: default with white (default is related to every state) and pressed with gray. The pressed state has 0x0020 precedence which is higher than the default state's 0x0000 precedence, so gray color will be used.
3. When the object is focused the same thing happens as in pressed state and red color will be used. (Focused state has higher precedence than default state).
4. When the object is focused and pressed both gray and red would work, but the pressed state has higher precedence than focused so gray color will be used.
5. It's possible to set e.g rose color for `LV_STATE_PRESSED | LV_STATE_FOCUSED`. In this case, this combined state has 0x0020 + 0x0002 = 0x0022 precedence, which higher than the pressed states precedence so rose color would be used.
6. When the object is in checked state there is no property to set the background color for this state. So in lack of a better option, the object remains white from the default state's property.

Some practical notes:

- The precedence (value) of states is quite intuitive and it's something the user would expect naturally. E.g. if an object is focused, the user still want to see if it's pressed, therefore pressed state has a higher precedence. If the focused state had higher precedence it would overwrite the pressed color.
- If you want to set a property for all state (e.g. red background color) just set it for the default state. If the object can't find a property for its current state it will fall back to the default state's property.
- Use ORed states to describe the properties for complex cases. (E.g. pressed + checked + focused)
- It might be a good idea to use different style elements for different states. For example, finding background colors for released, pressed, checked + pressed, focused, focused + pressed, focused + pressed + checked, etc states is quite difficult. Instead, for example, use the background color for pressed and checked states and indicate the focused state with a different border color.

4.3.2 Cascading styles

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

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

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

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

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

4.3.3 Inheritance

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

4.3.4 Parts

Objects can have *parts* which can have their own styles.

The following predefined parts exist in LVGL:

- `LV_PART_MAIN` A background like rectangle*/
- `LV_PART_SCROLLBAR` The scrollbar(s)
- `LV_PART_INDICATOR` Indicator, e.g. for slider, bar, switch, or the tick box of the checkbox
- `LV_PART_KNOB` Like a handle to grab to adjust the value*/
- `LV_PART_SELECTED` Indicate the currently selected option or section
- `LV_PART_ITEMS` Used if the widget has multiple similar elements (e.g. tabel cells)*/
- `LV_PART_TICKS` Ticks on scales e.g. for a chart or meter
- `LV_PART_CURSOR` Mark a specific place e.g. text area's or chart's cursor
- `LV_PART_CUSTOM_FIRST` Custom parts can be added from here.

For example a Slider has three parts:

- Background
- Indiactor
- Knob

It means the all three parts of the slider can have their own styles. See later how to add style styles to objects and parts.

4.3.5 Initialize styles and set/get properties

Styles are stored in `lv_style_t` variables. Style variables should be `static`, global or dynamically allocated. In other words they can not be local variables in functions which are destroyed when the function exists. Before using a style it should be initialized with `lv_style_init(&my_style)`. After initializing the style properties can be set or added to it.

Property set functions looks like this: `lv_style_set_<property_name>(&style, <value>);` For example:

```
static lv_style_t style_btn;
lv_style_init(&style_btn);
lv_style_set_bg_color(&style_btn, lv_color_grey());
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_color_red());
lv_style_set_bg_opa(&style_btn_red, LV_OPA_COVER);
```

To remove a property use:

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

To get a properties value from style:

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

`lv_style_value_t` has 3 fields:

- `num` for integer, boolean and opacity properties
- `color` for color properties
- `ptr` for pointer properties

To reset a style (free all its data) use

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

4.3.6 Add and remove styles to a widget

A style on its own not that useful. It should be assigned to an object to take its effect.

Add styles

To add a style to an object use `lv_obj_add_style(obj, &style, <selector>)`. `<selector>` is an OR-ed value of parts and state to which the style should be added. Some examples:

- `LV_PART_MAIN | LV_STATE_DEFAULT`
- `LV_STATE_PRESSED`: The main part in pressed state. `LV_PART_MAIN` can be omitted
- `LV_PART_SCROLLBAR`: The scrollbar part in the default state. `LV_STATE_DEFAULT` can be omitted.
- `LV_PART_SCROLLBAR | LV_STATE_SCROLLED`: The scrollbar part when the object is being scrolled
- `0` Same as `LV_PART_MAIN | LV_STATE_DEFAULT`.
- `LV_PART_INDICATOR | LV_STATE_PRESSED | LV_STATE_CHECKED` The indicator part when the object is pressed and checked at the same time.

Using `lv_obj_add_style`:

```
lv_obj_add_style(btn, &style_btn, 0);                                          /
↪*Default button style*/
lv_obj_add_style(btn, &btn_red, LV_STATE_PRESSED);  /*Overwrite only a some colors to␣
↪red when pressed*/
```

Remove styles

To remove all styles from an object use `lv_obj_remove_style_all(obj)`.

To remove specific styles use `lv_obj_remove_style(obj, style, selector)`. This function will remove `style` only if the `selector` matches with the `selector` used in `lv_obj_add_style`. `style` can be `NULL` to check only the `selector` and remove all matching styles. The `selector` can use the `LV_STATE_ANY` and `LV_PART_ANY` values to remove the style with any state or part.

Report style changes

If a style which is already assigned to object changes (i.e. a property is added or changed) the objects using that style should be notified. There are 3 options to do this:

1. If you know that the changed properties can be applied by a simple redraw (e.g. color or opacity changes) just call `lv_obj_invalidate(obj)` or `lv_obj_invalideate(lv_scr_act())`.
2. If more complex style properties were changed or added, and you know which object(s) are affected by that style call `lv_obj_refresh_style(obj, part, property)`. To refresh all parts and properties use `lv_obj_refresh_style(obj, LV_PART_ANY, LV_STYLE_PROP_ANY)`.
3. No make LVGL check all object whether they use the style and refresh them call `lv_obj_report_style_change(&style)`. If `style` is `NULL` all object's will be notified about the style change.

Get a property's value on an object

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

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

4.3.7 Local styles

Besides "normal" styles, the objects can store local styles too. This concept is similar to inline styles in CSS (e.g. `<div style="color:red">`) with some modification.

So local styles are like normal styles but they can't be shared among other objects. If used, local styles are allocated automatically, and freed when the object is deleted. They are usuful to add local customization to the object.

Unlike in CSS, in LVGL local styles can be assigned to states (*pseudo-classes*) and parts (pseudo-elements).

To set a local property use functions like `lv_obj_set_style_local_<property_name>(obj, <value>, <selector>);` For example:

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

4.3.8 Properties

For the full list of style properties click *here*.

Typical background properties

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

- Background
- Border
- Outline
- Shadow
- Padding
- Width and height transformation
- X and Y translation

4.3.9 Transitions

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

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

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

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

To describe a transition an `lv_transition_dsc_t` variable needs to initialized and added to a style:

```
/*Only its pointer is saved so must static, global or dynamically allocated */
static const lv_style_prop_t trans_props[] = {
                                                                                LV_
↪STYLE_BG_OPA, LV_STYLE_BG_COLOR,
                                                                                0,
↪ /*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);
```

4.3.10 Color filter

TODO

4.3.11 Themes

Themes are a collection of styles. If there is an active theme LVGL applies it on the every created widget. It gives a default appearance to UI which can be modified by adding further styles.

Every display can have a different theme. For example a colorful theme on a TFT and monochrome theme on a secondary monochrome display.

To set a theme for a display 2 steps are required:

1. Initialize a theme
2. Assign the initialized theme to a display.

Theme initialization functions can have different prototype. This example shows how to set the "default" theme:

```
lv_theme_t * th = lv_theme_default_init(display,  /*Use the DPI, size, etc from this␣
↪display*/
                                        LV_COLOR_PALETTE_BLUE, LV_COLOR_PALETTE_CYAN,␣
↪  /*Primary and secondary palette*/
                                        false,    /*Light or dark mode*/
                                        &lv_font_montserrat_10, &lv_font_montserrat_
↪14, &lv_font_montserrat_18); /*Small, normal, large fonts*/

lv_disp_set_theme(display, th); /*Assign the theme to the display*/
```

The themes can be enabled in `lv_conf.h`. If the default theme is enabled by `LV_USE_THEME_DEFAULT 1` LVGL automatically initializes and sets it when a display is created.

Extending themes

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

Here is an example about creating a custom theme based on the currently active theme.

```
/*Declare the style used in the theme*/
static lv_style_t style_btn;
...

/*Initialize the styles*/
lv_style_init(&style_btn);
lv_style_set_bg_color(&style_btn, lv_color_green());

/*Initialize the new theme from the current theme*/
lv_theme_t * th_act = lv_disp_get_theme(NULL);
static lv_theme_t th_new;
th_new = *th_act;

/*Set the parent theme ans the style applay callback for the new theme*/
lv_theme_set_parent(&th_new, th_act);
lv_theme_set_apply_cb(&th_new, new_theme_apply_cb);

/*Assign the new theme the the current display*/
lv_disp_set_theme(NULL, &th_new);

...

/*Will be called when the styles of the base theme are already added
  to add new styles*/
void new_theme_apply_cb(lv_theme_t * th, lv_obj_t * obj)
{
    if(lv_obj_check_type(obj, &lv_btn_class)) {
        lv_obj_add_style(obj, &style_btn, 0);
    }
}
```

4.3.12 Examples

4.3.13 API

Typedefs

typedef uint32_t **lv_style_selector_t**

Enums

enum **_lv_style_state_cmp_t**
Values:

enumerator **_LV_STYLE_STATE_CMP_SAME**

enumerator **_LV_STYLE_STATE_CMP_DIFF_REDRAW_MAIN**

enumerator **_LV_STYLE_STATE_CMP_DIFF_REDRAW_PART**

enumerator **_LV_STYLE_STATE_CMP_DIFF_DRAW_PAD**

enumerator **_LV_STYLE_STATE_CMP_DIFF_LAYOUT**

Functions

void **_lv_obj_style_init**(void)
Initialize the object related style manager module. Called by LVGL in `lv_init()`

void **lv_obj_add_style**(struct *_lv_obj_t* *obj, *lv_style_t* *style, *lv_style_selector_t* selector)

void **lv_obj_remove_style**(struct *_lv_obj_t* *obj, *lv_style_t* *style, *lv_style_selector_t* selector)

static inline void **lv_obj_remove_style_all**(struct *_lv_obj_t* *obj)
Remove all styles from an object

Parameters **obj** -- pointer to an object

void **lv_obj_report_style_change**(*lv_style_t* *style)
Notify all object if a style is modified

Parameters **style** -- pointer to a style. Only the objects with this style will be notified (NULL to notify all objects)

void **lv_obj_refresh_style**(struct *_lv_obj_t* *obj, *lv_part_t* part, *lv_style_prop_t* prop)
Notify an object and its children about its style is modified.

Parameters

- **obj** -- pointer to an object
- **part** -- the part whose style was changed. E.g. `LV_PART_ANY`, `LV_PART_MAIN`
- **prop** -- `LV_STYLE_PROP_ANY` or an `LV_STYLE_...` property. It is used to optimize what needs to be refreshed. `LV_STYLE_PROP_INV` to perform only a style cache update

void **lv_obj_enable_style_refresh**(bool en)

Enable or disable automatic style refreshing when a new style is added/removed to/from an object or any other style change happens.

Parameters **en** -- true: enable refreshing; false: disable refreshing

lv_style_value_t **lv_obj_get_style_prop**(const struct *_lv_obj_t* *obj, *lv_part_t* part, *lv_style_prop_t* prop)

Get the value of a style property. The current state of the object will be considered. Inherited properties will be inherited. If a property is not set a default value will be returned.

Parameters

- **obj** -- pointer to an object
- **part** -- a part from which the property should be get
- **prop** -- the property to get

Returns the value of the property. Should be read from the correct field of the `lv_style_value_t` according to the type of the property.

void **lv_obj_set_local_style_prop**(struct *_lv_obj_t* *obj, *lv_style_prop_t* prop, *lv_style_value_t* value, *lv_style_selector_t* selector)

Set local style property on an object's part and state.

Parameters

- **obj** -- pointer to an object
- **part** -- a part to which the property should be added
- **state** -- a state to which the property should be added
- **prop** -- the property
- **value** -- value of the property. The correct element should be set according to the type of the property

lv_res_t **lv_obj_get_local_style_prop**(struct *_lv_obj_t* *obj, *lv_style_prop_t* prop, *lv_style_value_t* *value, *lv_style_selector_t* selector)

bool **lv_obj_remove_local_style_prop**(struct *_lv_obj_t* *obj, *lv_style_prop_t* prop, *lv_style_selector_t* selector)

Remove a local style property from a part of an object with a given state.

Parameters

- **obj** -- pointer to an object
- **part** -- the part of the object which style property should be removed.
- **state** -- the state from which the property should be removed.
- **prop** -- a style property to remove.

Returns true the property was found and removed; false: the property was not found

void **_lv_obj_style_create_transition**(struct *_lv_obj_t* *obj, *lv_part_t* part, *lv_state_t* prev_state, *lv_state_t* new_state, const *_lv_obj_style_transition_dsc_t* *tr)

Used internally to create a style tarnsition

Parameters

- **obj** --
- **part** --

- **prev_state** --
- **new_state** --
- **tr** --

_lv_style_state_cmp_t **_lv_obj_style_state_compare**(struct *_lv_obj_t* *obj, *lv_state_t* state1, *lv_state_t* state2)

Used internally to compare the appearance of an object in 2 states

Parameters

- **obj** --
- **state1** --
- **state2** --

Returns

void **lv_obj_fade_in**(struct *_lv_obj_t* *obj, uint32_t time, uint32_t delay)

Fade in an an object and all its children.

Parameters

- **obj** -- the object to fade in
- **time** -- time of fade
- **delay** -- delay to start the animation

void **lv_obj_fade_out**(struct *_lv_obj_t* *obj, uint32_t time, uint32_t delay)

Fade out an an object and all its children.

Parameters

- **obj** -- the object to fade out
- **time** -- time of fade
- **delay** -- delay to start the animation

lv_state_t **lv_obj_style_get_selector_state**(*lv_style_selector_t* selector)

lv_part_t **lv_obj_style_get_selector_part**(*lv_style_selector_t* selector)

static inline void **lv_obj_set_style_pad_all**(struct *_lv_obj_t* *obj, lv_coord_t value, *lv_style_selector_t* selector)

static inline void **lv_obj_set_style_pad_hor**(struct *_lv_obj_t* *obj, lv_coord_t value, *lv_style_selector_t* selector)

static inline void **lv_obj_set_style_pad_ver**(struct *_lv_obj_t* *obj, lv_coord_t value, *lv_style_selector_t* selector)

static inline void **lv_obj_set_style_pad_gap**(struct *_lv_obj_t* *obj, lv_coord_t value, *lv_style_selector_t* selector)

static inline void **lv_obj_set_style_size**(struct *_lv_obj_t* *obj, lv_coord_t value, *lv_style_selector_t* selector)

struct **lv_obj_style_t**

Public Members

lv_style_t ***style**

uint32_t **selector**

uint32_t **is_local**

uint32_t **is_trans**

struct **_lv_obj_style_transition_dsc_t**

Public Members

uint16_t **time**

uint16_t **delay**

lv_style_selector_t **selector**

lv_style_prop_t **prop**

lv_anim_path_cb_t **path_cb**

void ***user_data**

Warning: doxygenfile: Cannot find file "lv_obj_style_dec.h

Typedefs

typedef uint8_t **lv_blend_mode_t**

typedef uint8_t **lv_text_decor_t**

typedef uint8_t **lv_border_side_t**

typedef uint8_t **lv_grad_dir_t**

typedef struct *_lv_style_transiton_t* **lv_style_transition_dsc_t**
Descriptor for style transitions

Enums

enum **[anonymous]**
Possible options how to blend opaque drawings

Values:

enumerator **LV_BLEND_MODE_NORMAL**
Simply mix according to the opacity value

enumerator **LV_BLEND_MODE_ADDITIVE**
Add the respective color channels

enumerator **LV_BLEND_MODE_SUBTRACTIVE**
Subtract the foreground from the background

enum **[anonymous]**
Some options to apply decorations on texts. 'OR'ed values can be used.

Values:

enumerator **LV_TEXT_DECOR_NONE**

enumerator **LV_TEXT_DECOR_UNDERLINE**

enumerator **LV_TEXT_DECOR_STRIKETHROUGH**

enum **[anonymous]**
Selects on which sides border should be drawn 'OR'ed values can be used.

Values:

enumerator **LV_BORDER_SIDE_NONE**

enumerator **LV_BORDER_SIDE_BOTTOM**

enumerator **LV_BORDER_SIDE_TOP**

enumerator **LV_BORDER_SIDE_LEFT**

enumerator **LV_BORDER_SIDE_RIGHT**

enumerator **LV_BORDER_SIDE_FULL**

enumerator **LV_BORDER_SIDE_INTERNAL**
FOR matrix-like objects (e.g. Button matrix)

enum **[anonymous]**
The direction of the gradient.

Values:

enumerator **LV_GRAD_DIR_NONE**
No gradient (the `grad_color` property is ignored)

enumerator **LV_GRAD_DIR_VER**
Vertical (top to bottom) gradient

enumerator **LV_GRAD_DIR_HOR**
Horizontal (left to right) gradient

enum **lv_style_prop_t**
Enumeration of all built in style properties

Values:

enumerator **LV_STYLE_PROP_INV**

enumerator **LV_STYLE_RADIUS**

enumerator **LV_STYLE_CLIP_CORNER**

enumerator **LV_STYLE_TRANSFORM_WIDTH**

enumerator **LV_STYLE_TRANSFORM_HEIGHT**

enumerator **LV_STYLE_TRANSLATE_X**

enumerator **LV_STYLE_TRANSLATE_Y**

enumerator **LV_STYLE_TRANSFORM_ZOOM**

enumerator **LV_STYLE_TRANSFORM_ANGLE**

enumerator **LV_STYLE_OPA**

enumerator **LV_STYLE_COLOR_FILTER_DSC**

enumerator **LV_STYLE_COLOR_FILTER_OPA**

enumerator **LV_STYLE_ANIM_TIME**

enumerator **LV_STYLE_ANIM_SPEED**

enumerator **LV_STYLE_TRANSITION**

enumerator **LV_STYLE_BLEND_MODE**

enumerator **LV_STYLE_PAD_TOP**

enumerator **LV_STYLE_PAD_BOTTOM**

enumerator **LV_STYLE_PAD_LEFT**

enumerator **LV_STYLE_PAD_RIGHT**

enumerator **LV_STYLE_PAD_ROW**

enumerator **LV_STYLE_PAD_COLUMN**

enumerator **LV_STYLE_WIDTH**

enumerator **LV_STYLE_MIN_WIDTH**

enumerator **LV_STYLE_MAX_WIDTH**

enumerator **LV_STYLE_HEIGHT**

enumerator **LV_STYLE_MIN_HEIGHT**

enumerator **LV_STYLE_MAX_HEIGHT**

enumerator **LV_STYLE_X**

enumerator **LV_STYLE_Y**

enumerator **LV_STYLE_LAYOUT**

enumerator **LV_STYLE_ALIGN**

enumerator **LV_STYLE_BG_COLOR**

enumerator **LV_STYLE_BG_COLOR_FILTERED**

enumerator **LV_STYLE_BG_OPA**

enumerator **LV_STYLE_BG_GRAD_COLOR**

enumerator **LV_STYLE_BG_GRAD_COLOR_FILTERED**

enumerator **LV_STYLE_BG_GRAD_DIR**

enumerator **LV_STYLE_BG_MAIN_STOP**

enumerator **LV_STYLE_BG_GRAD_STOP**

enumerator **LV_STYLE_BG_IMG_SRC**

enumerator **LV_STYLE_BG_IMG_OPA**

enumerator **LV_STYLE_BG_IMG_RECOLOR**

enumerator **LV_STYLE_BG_IMG_RECOLOR_FILTERED**

enumerator **LV_STYLE_BG_IMG_RECOLOR_OPA**

enumerator **LV_STYLE_BG_IMG_TILED**

enumerator **LV_STYLE_BORDER_COLOR**

enumerator **LV_STYLE_BORDER_COLOR_FILTERED**

enumerator **LV_STYLE_BORDER_OPA**

enumerator **LV_STYLE_BORDER_WIDTH**

enumerator **LV_STYLE_BORDER_SIDE**

enumerator **LV_STYLE_BORDER_POST**

enumerator **LV_STYLE_OUTLINE_WIDTH**

enumerator **LV_STYLE_OUTLINE_COLOR**

enumerator **LV_STYLE_OUTLINE_COLOR_FILTERED**

enumerator **LV_STYLE_OUTLINE_OPA**

enumerator **LV_STYLE_OUTLINE_PAD**

enumerator **LV_STYLE_SHADOW_WIDTH**

enumerator **LV_STYLE_SHADOW_OFS_X**

enumerator **LV_STYLE_SHADOW_OFS_Y**

enumerator **LV_STYLE_SHADOW_SPREAD**

enumerator **LV_STYLE_SHADOW_COLOR**

enumerator **LV_STYLE_SHADOW_COLOR_FILTERED**

enumerator **LV_STYLE_SHADOW_OPA**

enumerator **LV_STYLE_IMG_OPA**

enumerator **LV_STYLE_IMG_RECOLOR**

enumerator **LV_STYLE_IMG_RECOLOR_FILTERED**

enumerator **LV_STYLE_IMG_RECOLOR_OPA**

enumerator **LV_STYLE_LINE_WIDTH**

enumerator **LV_STYLE_LINE_DASH_WIDTH**

enumerator **LV_STYLE_LINE_DASH_GAP**

enumerator **LV_STYLE_LINE_ROUNDED**

enumerator **LV_STYLE_LINE_COLOR**

enumerator **LV_STYLE_LINE_COLOR_FILTERED**

enumerator **LV_STYLE_LINE_OPA**

enumerator **LV_STYLE_ARC_WIDTH**

enumerator **LV_STYLE_ARC_ROUNDED**

enumerator **LV_STYLE_ARC_COLOR**

enumerator **LV_STYLE_ARC_COLOR_FILTERED**

enumerator **LV_STYLE_ARC_OPA**

enumerator **LV_STYLE_ARC_IMG_SRC**

enumerator **LV_STYLE_TEXT_COLOR**

enumerator **LV_STYLE_TEXT_COLOR_FILTERED**

enumerator **LV_STYLE_TEXT_OPA**

enumerator **LV_STYLE_TEXT_FONT**

enumerator **LV_STYLE_TEXT_LETTER_SPACE**

enumerator **LV_STYLE_TEXT_LINE_SPACE**

enumerator **LV_STYLE_TEXT_DECOR**

enumerator **LV_STYLE_TEXT_ALIGN**

enumerator **_LV_STYLE_LAST_BUILT_IN_PROP**

enumerator **LV_STYLE_PROP_ANY**

Functions

LV_EXPORT_CONST_INT(LV_IMG_ZOOM_NONE)

void **lv_style_init**(*lv_style_t* *style)

Initialize a style

Note: Do not call `lv_style_init` on styles that are already have some properties because this function won't free the used memory just set a default state for the style. In other words be sure to initialize styles only once!

Parameters **style** -- pointer to a style to initialize

void **lv_style_reset**(*lv_style_t* *style)

Clear all properties from a style and free all allocated memories.

Parameters **style** -- pointer to a style

lv_style_prop_t **lv_style_register_prop**(void)

bool **lv_style_remove_prop**(*lv_style_t* *style, *lv_style_prop_t* prop)

Remove a property from a style

Parameters

- **style** -- pointer to a style

- **prop** -- a style property ORed with a state.

Returns true: the property was found and removed; false: the property wasn't found

void **lv_style_set_prop**(*lv_style_t* *style, *lv_style_prop_t* prop, *lv_style_value_t* value)

Set the value of property in a style. This function shouldn't be used directly by the user. Instead use `lv_style_set_<prop_name>()`. E.g. `lv_style_set_bg_color()`

Parameters

- **style** -- pointer to style
- **prop** -- the ID of a property (e.g. `LV_STLYE_BG_COLOR`)
- **value** -- *`lv_style_value_t`* variable in which a filed is set according to the type of `prop`

lv_res_t **lv_style_get_prop**(*lv_style_t* *style, *lv_style_prop_t* prop, *lv_style_value_t* *value)

Get the value of a property

Note: For performance reasons there are no sanity check on `style`

Parameters

- **style** -- pointer to a style
- **prop** -- the ID of a property
- **value** -- pointer to a *`lv_style_value_t`* variable to store the value

Returns LV_RES_INV: the property wsn't found in the style (`value` is unchanged) LV_RES_OK: the property was fond, and `value` is set accordingly

static inline lv_res_t **lv_style_get_prop_inlined**(*lv_style_t* *style, *lv_style_prop_t* prop, *lv_style_value_t* *value)

Get the value of a property

Note: For performance reasons there are no sanity check on `style`

Note: This function is the same as *lv_style_get_prop* but inlined. Use it only on performance critical places

Parameters

- **style** -- pointer to a style
- **prop** -- the ID of a property
- **value** -- pointer to a *`lv_style_value_t`* variable to store the value

Returns LV_RES_INV: the property wsn't found in the style (`value` is unchanged) LV_RES_OK: the property was fond, and `value` is set accordingly

void **lv_style_transition_dsc_init**(*lv_style_transition_dsc_t* *tr, const *lv_style_prop_t* props[], *lv_anim_path_cb_t* path_cb, uint32_t time, uint32_t delay, void *user_data)

lv_style_value_t **lv_style_prop_get_default**(*lv_style_prop_t* prop)
Get the default value of a property

Parameters **prop** -- the ID of a property

Returns the default value

bool **lv_style_is_empty**(const *lv_style_t* *style)
Checks if a style is empty (has no properties)

Parameters **style** -- pointer to a style

Returns

uint8_t **_lv_style_get_prop_group**(*lv_style_prop_t* prop)
Tell the group of a property. If the a property from a group is set in a style the (1 << group) bit of style->has_group is set. It allows early skipping the style if the property is not exists in the style at all.

Parameters **prop** -- a style property

Returns the group [0..7] 7 means all the custom properties with index > 112

static inline void **lv_style_set_pad_all**(*lv_style_t* *style, lv_coord_t value)

static inline void **lv_style_set_pad_hor**(*lv_style_t* *style, lv_coord_t value)

static inline void **lv_style_set_pad_ver**(*lv_style_t* *style, lv_coord_t value)

static inline void **lv_style_set_pad_gap**(*lv_style_t* *style, lv_coord_t value)

static inline void **lv_style_set_size**(*lv_style_t* *style, lv_coord_t value)

union **lv_style_value_t**
#include <lv_style.h> A common type to handle all the property types in the same way.

Public Members

int32_t **num**
Number integer number (opacity, enums, booleans or "normal" numbers)

const void ***ptr**
Constant pointers (font, cone text, etc)

lv_color_t **color**
Colors

struct **_lv_style_transiton_t**
#include <lv_style.h> Descriptor for style transitions

Public Members

const *lv_style_prop_t* ***props**
An array with the properties to animate.

void ***user_data**
A custom user data that will be passed to the animation's user_data

lv_anim_path_cb_t **path_xcb**
A path for the animation.

uint32_t **time**
Duration of the transition in [ms]

uint32_t **delay**
Delay before the transition in [ms]

struct **lv_style_const_prop_t**
#include <lv_style.h> Descriptor of a constant style property.

Public Members

lv_style_prop_t **prop**

lv_style_value_t **value**

struct **lv_style_t**
#include <lv_style.h> Descriptor of a style (a collection of properties and values).

Public Members

uint32_t **sentinel**

lv_style_value_t **value1**

uint8_t ***values_and_props**

const *lv_style_const_prop_t* ***const_props**

union *lv_style_t*::[anonymous] **v_p**

uint16_t **prop1**

uint16_t **is_const**

uint8_t **has_group**

uint8_t **prop_cnt**

Warning: doxygenfile: Cannot find file "lv_style_dec.h

Typedefs

typedef void (*`lv_theme_apply_cb_t`)(struct *_lv_theme_t*, *lv_obj_t*)

typedef struct *_lv_theme_t* `lv_theme_t`

Functions

lv_theme_t *`lv_theme_get_from_obj`(*lv_obj_t* *obj)

Get the theme assigned to the display of the object

Parameters `obj` -- pointer to object

Returns the theme of the object's display (can be NULL)

void `lv_theme_apply`(*lv_obj_t* *obj)

Apply the active theme on an object

Parameters `obj` -- pointer to an object

void `lv_theme_set_parent`(*lv_theme_t* *new_theme, *lv_theme_t* *parent)

Set a base theme for a theme. The styles from the base them will be added before the styles of the current theme. Arbitrary long chain of themes can be created by setting base themes.

Parameters

- `new_theme` -- pointer to theme which base should be set
- `parent` -- pointer to the base theme

void `lv_theme_set_apply_cb`(*lv_theme_t* *theme, *lv_theme_apply_cb_t* apply_cb)

Set an apply callback for a theme. The apply callback is used to add styles to different objects

Parameters

- `theme` -- pointer to theme which callback should be set
- `apply_cb` -- pointer to the callback

const lv_font_t *`lv_theme_get_font_small`(*lv_obj_t* *obj)

Get the small font of the theme

Returns pointer to the font

const lv_font_t *`lv_theme_get_font_normal`(*lv_obj_t* *obj)

Get the normal font of the theme

Returns pointer to the font

const lv_font_t *`lv_theme_get_font_large`(*lv_obj_t* *obj)

Get the subtitle font of the theme

Returns pointer to the font

lv_color_t `lv_theme_get_color_primary`(*lv_obj_t* *obj)

Get the primary color of the theme

Returns the color

lv_color_t `lv_theme_get_color_secondary`(*lv_obj_t* *obj)

Get the secondary color of the theme

Returns the color

struct **_lv_theme_t**

Public Members

lv_theme_apply_cb_t **apply_cb**

struct *_lv_theme_t* ***parent**
Apply the current theme's style on top of this theme.

void ***user_data**

struct *_lv_disp_t* ***disp**

lv_color_t **color_primary**

lv_color_t **color_secondary**

const lv_font_t ***font_small**

const lv_font_t ***font_normal**

const lv_font_t ***font_large**

uint32_t **flags**

4.4 Style properties

4.4.1 Miscellaneous

TODO

radius

Set the radius on every corner. The value is interpreted in pixel (>= 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`.

transform_width

Make the object wider on both sides with this value. Pixel and percentage (with `lv_pct(x)`) values can be used. Percentage values are relative to the object's width.

transform_height

Make the object higher on both sides with this value. Pixel and percentage (with `lv_pct(x)`) values can be used. Percentage values are relative to the object's height.

translate_x

Move the object with this value in X direction. Applied after layouts, aligns and other positioning. Pixel and percentage (with `lv_pct(x)`) values can be used. Percentage values are relative to the object's width.

translate_y

Move the object with this value in Y direction. Applied after layouts, aligns and other positioning. Pixel and percentage (with `lv_pct(x)`) values can be used. Percentage values are relative to the object's height.

transform_zoom

Zoom image-like objects. Multiplied with the zoom set on the object. The value 256 (or `LV_IMG_ZOOM_NONE`) means normal size, 128 half size, 512 double size, and so on

transform_angle

Rotate image-like objects. Added to the rotation set on the object. The value is interpreted in 0.1 degree unit. E.g. 45 deg. = 450

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 possibel values are `LV_BLEND_MODE_NORMAL/ADDITIVE/SUBTRACTIVE`

4.4.2 Padding

TODO

pad_top

Sets the padding on the top. It makes the content area smaller in this direction.

pad_bottom

Sets the padding on the bottom. It makes the content area smaller in this direction.

pad_left

Sets the padding on the left. It makes the content area smaller in this direction.

pad_right

Sets the padding on the right. It makes the content area smaller in this direction.

pad_row

Sets the padding between the rows. Used by the layouts.

pad_column

Sets the padding between the columns. Used by the layouts.

4.4.3 Size and position

TODO

width

Sets the width of object. Pixel, percentage and `LV_SIZE_CONTENT` values can be used. Percentage values are relative to the width of the parent's content area.

min_width

Sets a minimal width. Pixel and percentage values can be used. Percentage values are relative to the width of the parent's content area.

max_width

Sets a maximal width. Pixel and percentage values can be used. Percentage values are relative to the width of the parent's content area.

height

Sets the height of object. Pixel, percentage and `LV_SIZE_CONTENT` can be used. Percentage values are relative to the height of the parent's content area.

min_height

Sets a minimal height. Pixel and percentage values can be used. Percentage values are relative to the width of the parent's content area.

max_height

Sets a maximal height. Pixel and percentage values can be used. Percentage values are relative to the height of the parent's content area.

x

Set the X coordinate of the object considering the set `align`. Pixel and percentage values can be used. Percentage values are relative to the width of the parent's content area.

y

Set the Y coordinate of the object considering the set `align`. Pixel and percentage values can be used. Percentage values are relative to the height of the parent's content area.

align

Set the alignment which tells from which point of the parent the X and Y coordinates should be interpreted. The possible values are: `LV_ALIGN_TOP_LEFT/MID/RIGHT`, `LV_ALIGN_BOTTOM_LEFT/MID/RIGHT`, `LV_ALIGN_LEFT/RIGHT_MID`, `LV_ALIGN_CENTER`

layout

Set the layout if 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.

4.4.4 Background

TODO

bg_color

Set the background color of the object.

bg_opa

Set the opacity of the background. 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.

bg_grad_color

Set the gradient color of the background. Used only if `grad_dir` is not `LV_GRAD_DIR_NONE`

bg_grad_dir

Set the direction of the gradient of the background. The possible values are `LV_GRAD_DIR_NONE/HOR/VER`.

bg_main_stop

Set the point from which the background color should start for gradients. 0 means to top/left side, 255 the bottom/right side, 128 the center, and so on

bg_grad_stop

Set the point from which the background's gradient color should start. 0 means to top/left side, 255 the bottom/right side, 128 the center, and so on

bg_img_src

Set a background image. Can be a pointer to `lv_img_dsc_t`, a path to a file or an `LV_SYMBOL_...`

bg_img_opa

Set the opacity of the background 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.

bg_img_recolor

Set a color to mix to the background image.

bg_img_recolor_opa

Set the intensity of background image recoloring. Value 0, `LV_OPA_0` or `LV_OPA_TRANSP` means no mixing, 256, `LV_OPA_100` or `LV_OPA_COVER` means full recoloring, other values or LV_OPA_10, LV_OPA_20, etc are interpreted proportionally.

bg_img_tiled

If enbaled the background image will be tiled. The possible values are `true` or `false`.

4.4.5 Border

TODO

border_color

Set the color of the border

border_opa

Set the opcitiy of the border. 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.

border_width

Set hte width of the border. Only pixel values can be used.

border_side

Set ony which side(s) the border should be drawn. The possible values are `LV_BORDER_SIDE_NONE/TOP/BOTTOM/LEFT/RIGHT/INTERNAL`. OR-ed calues an be used as well, e.g. `LV_BORDER_SIDE_TOP | LV_BORDER_SIDE_LEFT`.

border_post

Sets wheter the the border should be drawn before or after the children ar drawn. `true`: after children, `false`: before children

4.4.6 Text

TODO

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

4.4.7 Image

TODO

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 mixt 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.

4.4.8 Outline

TODO

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.

4.4.9 Shadow

TODO

shadow_width

Set the width of the shadow in pixels. The value should be >= 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 calcuation to use a larger or smaller rectangle as base. The value can be in pixel t 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.

4.4.10 Line

TODO

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`: perpandicular line ending

line_color

Set the color fo the lines.

line_opa

Set the opacity of the lines.

4.4.11 Arc

TODO

arc_width

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

arc_rounded

Make the end points of the arcs rounded. `true`: rounded, `false`: perpandicular 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

4.5 Scroll

4.5.1 Overview

In LVGL scrolling works very intuitively: if an object is out of its parent content area (the size without paddings), the parent becomes scrollable and scrollbar(s) will appear. That's it.

Any object can be scrollable including `lv_obj_t`, `lv_img`, `lv_btn`, `lv_meter`, etc

The obejct can be scrolled either horizontally or vertically at a time, that is diagonal scrolling is not possible.

Scrollbar

Mode

The scrollbars are displayed according to the set `mode`. The following `mode`s exist:

- `LV_SCROLLBAR_MODE_OFF` Never show the scrollbars
- `LV_SCROLLBAR_MODE_ON` Always show the scrollbars
- `LV_SCROLLBAR_MODE_ACTIVE` Show scroll bars while object is being scrolled
- `LV_SCROLLBAR_MODE_AUTO` Show scroll bars when the content is large enough to be scrolled

`lv_obj_set_scrollbar_mode(obj, LV_SCROLLBAR_MODE_...)` set the scrollbar mode on an object.

Styling

The scrollbars have its own dedicated part, called `LV_PART_SCROLLBAR`. For example a scrollbar can turned to red like this:

```
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);
```

The object goes to `LV_STATE_SCROLLED` state while it's being scrolled. It allows adding different style to the scrollbar or the object itself when scrolled. This code makes the scrollbar blue when the object is scrolled:

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

...

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

Events

The following events are related to scrolling:

- `LV_EVENT_SCROLL_BEGIN` Scrolling begins
- `LV_EVENT_SCROLL_END` Scrolling ends
- `LV_EVENT_SCROLL` Scroll happened. Triggered on every position change. Scroll events

4.5.2 Basic example

TODO

4.5.3 Features of scrolling

Besides managing "normal" scrolling there are many interesting and useful additional features too.

Scrollable

It's possible to make an object non-scrollable with `lv_obj_add_flag(obj, LV_OBJ_FLAG_SCROLLABLE)`.

Non-scrollable object can still propagate the scrolling (chain) to the parents.

The direction in which scrolling can happen can be controlled by `lv_obj_set_scroll_dir(obj, LV_DIR_...)`. The following values are possible for the direction:

- `LV_DIR_TOP` only scroll up
- `LV_DIR_LEFT` only scroll left

- `LV_DIR_BOTTOM` only scroll down
- `LV_DIR_RIGHT` only scroll right
- `LV_DIR_HOR` only scroll horizontally
- `LV_DIR_TOP` only scroll vertically
- `LV_DIR_ALL` scroll any directions

OR-ed values are also possible. E.g. `LV_DIR_TOP | LV_DIR_LEFT`.

Scroll chain

If an object can't be scrolled further (e.g. it's content has reached the bottom most position) the scrolling is propagated to it's parent. If the parent an be scrolled in that direction than it will be scrolled instead. It goes to the grad parent and grand grandparents too.

The propagation on scrolling in called "scroll chaining" and it can be enabled/disabled with the `LV_OBJ_FLAG_SCROLL_CHAIN` flag. If chaining is disabled the propagation stops on the object and the parent(s) won't be scrolled.

Scroll momentum

When the user scrolls an object and releases it LVGL can emulate a momentum for the scrolling. It's like to object were thrown and the scrolling slows down smoothly.

The scroll momentum can be enabled/disabled with the `LV_OBJ_FLAG_SCROLL_MOMENTUM` flag.

Elastic scroll

Normally the content can't be scrolled inside the object. That is the top side of the content can't be below the top side of the object.

However, with `LV_OBJ_FLAG_SCROLL_ELASTIC` a fancy effect can be added when the user "over-scrolls" the content. The scrolling slows down, and the content can be scrolled inside the object. When the object is releases the content is scrolled in it will be animated back to the valid position.

Snaping

The children of an object can be snapped according to specific rules when scrolling ends. Children can be made snapable individually with the `LV_OBJ_FLAG_SNAPABLE` flag. The object can align the snapped children in 4 ways:

- `LV_SCROLL_SNAP_NONE` Snapping is disabled. (default)
- `LV_SCROLL_SNAP_START` Align the children to the left/top side of the scrolled object
- `LV_SCROLL_SNAP_END` Align the children to the right/bottom side of the scrolled object
- `LV_SCROLL_SNAP_CENTER` Align the children to the center of the scrolled object

The alignment can be set with `lv_obj_set_scroll_snap_x/y(obj, LV_SCROLL_SNAP_...)`:

Under the hood the followings happen

1. User scrolls an object and releases the screen
2. LVGL calculates where would the scroll end considering scroll momentum

3. LVGL finds the nearest scroll point
4. LVGL scrolls the snap point with an animation

Scroll one

The "scroll one" feature tells LVGL to allow scrolling only one snapable children at a time. So it requires to make the children snapable and and set a scroll snap alignment different from `LV_SCROLL_SNAP_NONE`.

This feature can be enabled by the `LV_OBJ_FLAG_SCROLL_ONE` flag.

Scroll on focus

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

4.5.4 Scroll manually

The following API functions allow to manually scroll 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 left side

4.5.5 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 tell the size of the 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.

It means not only the children can make an object scrollable but a larger self size 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 neesd to be calculated now
  if(p->x >= 0) {
    p->x = 200;        //Set or calculate the self width
  }

  if(p->x >= 0) {
```

(continues on next page)

(continued from previous page)

```
    p->y = 50;        //Set or calculate the self height
  }
}
```

4.5.6 Examples

4.6 Layers

4.6.1 Order of creation

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

For example, assume we added a button to a parent object named button1 and then another button named button2. Then button1 (with its child object(s)) will be in the background and can be covered by button2 and its children.


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

/*Create 2 buttons*/
lv_obj_t * btn1 = lv_btn_create(scr, NULL);         /*Create a button on the screen*/
lv_btn_set_fit(btn1, true, true);                   /*Enable to automatically set the␣
↪size according to the content*/
lv_obj_set_pos(btn1, 60, 40);                         /*Set the position of the␣
↪button*/

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

/*Add labels to the buttons*/
lv_obj_t * label1 = lv_label_create(btn1, NULL);        /*Create a label on the first␣
↪button*/
```

(continues on next page)

(continued from previous page)

```
lv_label_set_text(label1, "Button 1");                    /*Set the text of the label*/

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

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

4.6.2 Bring to the foreground

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

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

4.6.3 Top and sys layers

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

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

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

The `layer_sys` is also used for a similar purpose on LVGL. For example, it places the mouse cursor above all layers to be sure it's always visible.

4.7 Events

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

- is clicked
- is scrolled
- its value has changed
- redrawn, etc.

4.7.1 Add events to the object

The user can assign callback functions to an object to see its events. In practice, it looks like this:

```
lv_obj_t * btn = lv_btn_create(lv_scr_act());
lv_obj_add_event_cb(btn, my_event_cb, LV_EVENT_CLICKED, NULL);   /*Assign an event␣
↪callback*/

...

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

In the example `LV_EVENT_CLICKED` means that only the click event will call `my_event_cb`. See the *list of event codes* for all the options. `LV_EVENT_ALL` can be used to receive all the events.

The last parameter of `lv_obj_add_event_cb` is a pointer to any custom data that will be available in the event. It will be described later in more detail.

More events can be added to an object, like this:

```
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*/
```

Even the same event callback can be used on an object with different `user_data`. For example:

```
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);
```

The events will be called in the order as they were added.

More objects can use the same *event callback*.

4.7.2 Remove event(s) from an object

Events can be removed from an object with the `lv_obj_remove_event_cb(obj, event_cb)` function or `lv_obj_remove_event_dsc(obj, event_dsc)`. `event_dsc` is a pointer returned by `lv_obj_add_event_cb`.

4.7.3 Event codes

The event codes can be grouped into these categories:

- Input device events
- Drawing events
- Other events
- Special events
- Custom events

All objects (such as Buttons/Labels/Sliders etc.) regardless their type receive the *Input device*, *Drawing* and *Other* events.

However the *Special events* are specific to a particular widget type. See the *widgets' documentation* to learn when they are sent,

Custom events are added by the user and therefore these are never sent by LVGL.

The following event codes exist:

Input device events

- `LV_EVENT_PRESSED` The object has been pressed
- `LV_EVENT_PRESSING` The object is being pressed (called continuously while pressing)
- `LV_EVENT_PRESS_LOST` The object is still being pressed but slid cursor/finger off of the object
- `LV_EVENT_SHORT_CLICKED` The object was pressed for a short period of time, then released it. Not called if scrolled.
- `LV_EVENT_LONG_PRESSED` Object has been pressed for at least the `long_press_time` specified in the input device driver. Not called if scrolled.
- `LV_EVENT_LONG_PRESSED_REPEAT` Called after `long_press_time` in every `long_press_repeat_time` ms. Not called if scrolled.
- `LV_EVENT_CLICKED` Called on release if the object not scrolled (regardless to long press)
- `LV_EVENT_RELEASED` Called in every cases when the object has been released
- `LV_EVENT_SCROLL_BEGIN` Scrolling begins
- `LV_EVENT_SCROLL_END` Scrolling ends
- `LV_EVENT_SCROLL` The object was scrolled
- `LV_EVENT_GESTURE` A gesture is detected. Get the gesture with `lv_indev_get_gesture_dir(lv_indev_get_act());`
- `LV_EVENT_KEY` A key is sent to the object. Get the key with `lv_indev_get_key(lv_indev_get_act());`
- `LV_EVENT_FOCUSED` The object is focused
- `LV_EVENT_DEFOCUSED` The object is defocused
- `LV_EVENT_LEAVE` The object is defocused but still selected
- `LV_EVENT_HIT_TEST` Perform advanced hit-testing

Drawing events

- `LV_EVENT_COVER_CHECK` Check if the object fully covers an area. The event parameter is `lv_cover_check_info_t *`.
- `LV_EVENT_REFR_EXT_DRAW_SIZE` Get the required extra draw area around the object (e.g. for shadow). The event parameter is `lv_coord_t *` to store the size. Overwrite it only 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*.

Other events

- `LV_EVENT_DELETE` Object is being deleted
- `LV_EVENT_CHILD_CHANGED` Child was removed/added
- `LV_EVENT_SIZE_CHANGED` Object coordinates/size have changed
- `LV_EVENT_STYLE_CHANGED` Object's style has changed
- `LV_EVENT_BASE_DIR_CHANGED` The base dir has changed
- `LV_EVENT_GET_SELF_SIZE` Get the internal size of a widget

Special events

- `LV_EVENT_VALUE_CHANGED` The object's value has changed (i.e. slider moved)
- `LV_EVENT_INSERT` A text is being inserted to the object. The event data is `char *` being inserted.
- `LV_EVENT_REFRESH` Notify the object to refresh something on it (for the user)
- `LV_EVENT_READY` A process has finished
- `LV_EVENT_CANCEL` A process has been canceled

Custom events

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

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

4.7.4 Sending events

To manually send events to an object, use `lv_event_send(obj, <EVENT_CODE> &some_data)`.

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

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

Refresh event

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

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

4.7.5 Fields of lv_event_t

`lv_event_t` is the only parameter passed to event callback and it contains all the data about the event. The following values can be get from it:

- `lv_event_get_code(e)` get the event code
- `lv_event_get_target(e)` get the object to which the event is sent
- `lv_event_get_original_target(e)` get the object to which the event is sent originally sent (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`

4.7.6 Event bubbling

If `lv_obj_add_flag(obj, LV_OBJ_FLAG_EVENT_BUBBLE)` is enabled all events will be sent to the object's parent too. If the parent also has `LV_OBJ_FLAG_EVENT_BUBBLE` enabled the event will be sent to its parent too, 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.

4.8 Input devices

An input device usually means:

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

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

4.8.1 Pointers

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

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

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

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

4.8.2 Keypad and encoder

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

Groups

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

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

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

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

Keys

There are some predefined keys which have special meaning:

- **LV_KEY_NEXT** Focus on the next object
- **LV_KEY_PREV** Focus on the previous object
- **LV_KEY_ENTER** Triggers `LV_EVENT_PRESSED/CLICKED/LONG_PRESSED` etc. events
- **LV_KEY_UP** Increase value or move upwards
- **LV_KEY_DOWN** Decrease value or move downwards
- **LV_KEY_RIGHT** Increase value or move the the right
- **LV_KEY_LEFT** Decrease value or move the the left
- **LV_KEY_ESC** Close or exit (E.g. close a *Drop down list*)

- **LV_KEY_DEL** Delete (E.g. a character on the right in a *Text area*)
- **LV_KEY_BACKSPACE** Delete a character on the left (E.g. in a *Text area*)
- **LV_KEY_HOME** Go to the beginning/top (E.g. in a *Text area*)
- **LV_KEY_END** Go to the end (E.g. in a *Text area*))

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

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

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

Edit and navigate mode

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

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

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

Styling

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

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

For a more detaild description read the Style section.

4.8.3 API

Input device

Functions

void **lv_indev_read_timer_cb**(*lv_timer_t* *timer)

Called periodically to read the input devices

Parameters **param** -- pointer to and input device to read

void **lv_indev_enable**(*lv_indev_t* *indev, bool en)

lv_indev_t ***lv_indev_get_act**(void)

Get the currently processed input device. Can be used in action functions too.

Returns pointer to the currently processed input device or NULL if no input device processing right now

lv_indev_type_t **lv_indev_get_type**(const *lv_indev_t* *indev)
Get the type of an input device

Parameters **indev** -- pointer to an input device

Returns the type of the input device from `lv_hal_indev_type_t` (`LV_INDEV_TYPE_...`)

void **lv_indev_reset**(*lv_indev_t* *indev, *lv_obj_t* *obj)
Reset one or all input devices

Parameters

- **indev** -- pointer to an input device to reset or NULL to reset all of them
- **obj** -- pointer to an object which triggers the reset.

void **lv_indev_reset_long_press**(*lv_indev_t* *indev)
Reset the long press state of an input device

Parameters **indev** -- pointer to an input device

void **lv_indev_set_cursor**(*lv_indev_t* *indev, *lv_obj_t* *cur_obj)
Set a cursor for a pointer input device (for LV_INPUT_TYPE_POINTER and LV_INPUT_TYPE_BUTTON)

Parameters

- **indev** -- pointer to an input device
- **cur_obj** -- pointer to an object to be used as cursor

void **lv_indev_set_group**(*lv_indev_t* *indev, *lv_group_t* *group)
Set a destination group for a keypad input device (for LV_INDEV_TYPE_KEYPAD)

Parameters

- **indev** -- pointer to an input device
- **group** -- point to a group

void **lv_indev_set_button_points**(*lv_indev_t* *indev, const lv_point_t points[])
Set the an array of points for LV_INDEV_TYPE_BUTTON. These points will be assigned to the buttons to press a specific point on the screen

Parameters

- **indev** -- pointer to an input device
- **group** -- point to a group

void **lv_indev_get_point**(const *lv_indev_t* *indev, lv_point_t *point)
Get the last point of an input device (for LV_INDEV_TYPE_POINTER and LV_INDEV_TYPE_BUTTON)

Parameters

- **indev** -- pointer to an input device
- **point** -- pointer to a point to store the result

lv_dir_t **lv_indev_get_gesture_dir**(const *lv_indev_t* *indev)
Get the current gesture direct

Parameters **indev** -- pointer to an input device

Returns current gesture direct

uint32_t **lv_indev_get_key**(const *lv_indev_t* *indev)
Get the last pressed key of an input device (for LV_INDEV_TYPE_KEYPAD)

Parameters **indev** -- pointer to an input device

Returns the last pressed key (0 on error)

lv_dir_t **lv_indev_get_scroll_dir**(const *lv_indev_t* *indev)

Check the current scroll direction of an input device (for LV_INDEV_TYPE_POINTER and LV_INDEV_TYPE_BUTTON)

Parameters **indev** -- pointer to an input device

Returns LV_DIR_NONE: no scrolling now LV_DIR_HOR/VER

lv_obj_t ***lv_indev_get_scroll_obj**(const *lv_indev_t* *indev)

Get the currently scrolled object (for LV_INDEV_TYPE_POINTER and LV_INDEV_TYPE_BUTTON)

Parameters **indev** -- pointer to an input device

Returns pointer to the currently scrolled object or NULL if no scrolling by this indev

void **lv_indev_get_vect**(const *lv_indev_t* *indev, lv_point_t *point)

Get the movement vector of an input device (for LV_INDEV_TYPE_POINTER and LV_INDEV_TYPE_BUTTON)

Parameters

- **indev** -- pointer to an input device
- **point** -- pointer to a point to store the types.pointer.vector

void **lv_indev_wait_release**(*lv_indev_t* *indev)

Do nothing until the next release

Parameters **indev** -- pointer to an input device

lv_obj_t ***lv_indev_get_obj_act**(void)

Gets a pointer to the currently active object in the currently processed input device.

Returns pointer to currently active object or NULL if no active object

lv_timer_t ***lv_indev_get_read_timer**(*lv_disp_t* *indev)

Get a pointer to the indev read timer to modify its parameters with `lv_timer_...` functions.

Parameters **indev** -- pointer to an input device

Returns pointer to the indev read refresher timer. (NULL on error)

lv_obj_t ***lv_indev_search_obj**(*lv_obj_t* *obj, lv_point_t *point)

Search the most top, clickable object by a point

Parameters

- **obj** -- pointer to a start object, typically the screen
- **point** -- pointer to a point for searching the most top child

Returns pointer to the found object or NULL if there was no suitable object

Groups

Typedefs

typedef uint8_t **lv_key_t**

typedef void (*__lv_group_focus_cb_t__)(struct *_lv_group_t**)

typedef struct *_lv_group_t* **lv_group_t**

Groups can be used to logically hold objects so that they can be individually focused. They are NOT for laying out objects on a screen (try `lv_cont` for that).

typedef uint8_t **lv_group_refocus_policy_t**

Enums

enum **[anonymous]**

Values:

enumerator **LV_KEY_UP**

enumerator **LV_KEY_DOWN**

enumerator **LV_KEY_RIGHT**

enumerator **LV_KEY_LEFT**

enumerator **LV_KEY_ESC**

enumerator **LV_KEY_DEL**

enumerator **LV_KEY_BACKSPACE**

enumerator **LV_KEY_ENTER**

enumerator **LV_KEY_NEXT**

enumerator **LV_KEY_PREV**

enumerator **LV_KEY_HOME**

enumerator **LV_KEY_END**

enum **[anonymous]**

Values:

enumerator **LV_GROUP_REFOCUS_POLICY_NEXT**

enumerator **LV_GROUP_REFOCUS_POLICY_PREV**

Functions

void **_lv_group_init**(void)
Init. the group module

Remark Internal function, do not call directly.

lv_group_t ***lv_group_create**(void)
Create a new object group

Returns pointer to the new object group

void **lv_group_del**(*lv_group_t* *group)
Delete a group object

Parameters **group** -- pointer to a group

void **lv_group_set_default**(*lv_group_t* *group)
Set a default group. New object are added to this group if it's enabled in their class with `add_to_def_group = true`

Parameters **group** -- pointer to a group (can be `NULL`)

lv_group_t ***lv_group_get_default**(void)
Get the default group

Returns pointer to the default group

void **lv_group_add_obj**(*lv_group_t* *group, struct *_lv_obj_t* *obj)
Add an object to a group

Parameters

- **group** -- pointer to a group
- **obj** -- pointer to an object to add

void **lv_group_remove_obj**(struct *_lv_obj_t* *obj)
Remove an object from its group

Parameters **obj** -- pointer to an object to remove

void **lv_group_remove_all_objs**(*lv_group_t* *group)
Remove all objects from a group

Parameters **group** -- pointer to a group

void **lv_group_focus_obj**(struct *_lv_obj_t* *obj)
Focus on an object (defocus the current)

Parameters **obj** -- pointer to an object to focus on

void **lv_group_focus_next**(*lv_group_t* *group)
Focus the next object in a group (defocus the current)

Parameters **group** -- pointer to a group

void **lv_group_focus_prev**(*lv_group_t* *group)
Focus the previous object in a group (defocus the current)

Parameters **group** -- pointer to a group

void **lv_group_focus_freeze**(*lv_group_t* *group, bool en)
Do not let to change the focus from the current object

Parameters

- **group** -- pointer to a group
- **en** -- true: freeze, false: release freezing (normal mode)

lv_res_t **lv_group_send_data**(*lv_group_t* *group, uint32_t c)
Send a control character to the focuses object of a group

Parameters

- **group** -- pointer to a group
- **c** -- a character (use LV_KEY_.. to navigate)

Returns result of focused object in group.

void **lv_group_set_focus_cb**(*lv_group_t* *group, *lv_group_focus_cb_t* focus_cb)
Set a function for a group which will be called when a new object is focused

Parameters

- **group** -- pointer to a group
- **focus_cb** -- the call back function or NULL if unused

void **lv_group_set_refocus_policy**(*lv_group_t* *group, *lv_group_refocus_policy_t* policy)
Set whether the next or previous item in a group is focused if the currently focused obj is deleted.

Parameters

- **group** -- pointer to a group
- **policy** -- new refocus policy enum

void **lv_group_set_editing**(*lv_group_t* *group, bool edit)
Manually set the current mode (edit or navigate).

Parameters

- **group** -- pointer to group
- **edit** -- true: edit mode; false: navigate mode

void **lv_group_set_wrap**(*lv_group_t* *group, bool en)
Set whether focus next/prev will allow wrapping from first->last or last->first object.

Parameters

- **group** -- pointer to group
- **en** -- true: wrapping enabled; false: wrapping disabled

struct *_lv_obj_t* ***lv_group_get_focused**(const *lv_group_t* *group)
Get the focused object or NULL if there isn't one

Parameters **group** -- pointer to a group

Returns pointer to the focused object

lv_group_focus_cb_t **lv_group_get_focus_cb**(const *lv_group_t* *group)
Get the focus callback function of a group

Parameters **group** -- pointer to a group

Returns the call back function or NULL if not set

bool **lv_group_get_editing**(const *lv_group_t* *group)
Get the current mode (edit or navigate).

Parameters **group** -- pointer to group

Returns true: edit mode; false: navigate mode

bool **lv_group_get_wrap**(*lv_group_t* *group)

Get whether focus next/prev will allow wrapping from first->last or last->first object.

Parameters

- **group** -- pointer to group
- **en** -- true: wrapping enabled; false: wrapping disabled

struct **_lv_group_t**

#include <lv_group.h> Groups can be used to logically hold objects so that they can be individually focused. They are NOT for laying out objects on a screen (try `lv_cont` for that).

Public Members

lv_ll_t **obj_ll**

Linked list to store the objects in the group

struct *_lv_obj_t* ****obj_focus**

The object in focus

lv_group_focus_cb_t **focus_cb**

A function to call when a new object is focused (optional)

void ***user_data**

uint8_t **frozen**

1: can't focus to new object

uint8_t **editing**

1: Edit mode, 0: Navigate mode

uint8_t **refocus_policy**

1: Focus prev if focused on deletion. 0: Focus next if focused on deletion.

uint8_t **wrap**

1: Focus next/prev can wrap at end of list. 0: Focus next/prev stops at end of list.

4.9 Displays

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

4.9.1 Multiple display support

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

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

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

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

Using only one display

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

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

Mirror display

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

Split image

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

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

4.9.2 Screens

Every display has each set of Screens and the object on the screens.

Be sure not to confuse displays and screens:

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

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

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

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

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

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

Transparent screens

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

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

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

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

- Enable `LV_COLOR_SCREEN_TRANSP` in `lv_conf.h`
- Be sure to use `LV_COLOR_DEPTH 32`
- Set the screens 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);`

4.9.3 Features of displays

Inactivity

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

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

Background

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

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

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

4.9.4 Colors

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

The following variable types are defined by the color module:

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

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

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

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

Convert color

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

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

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

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

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

lv_color32_t c24;
c32.full = lv_color_to32(c);        /*Give a 32 bit number with the converted color*/
```

Swap 16 colors

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

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

Create and mix colors

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

Colors can be created from HEX codes too: `my_color = lv_color_hex(0x288ACF)` or `my_color = lv_folro_hex3(0x28C)`.

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

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

Opacity

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

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

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

4.9.5 API

Display

Enums

enum **`lv_scr_load_anim_t`**
Values:

enumerator **`LV_SCR_LOAD_ANIM_NONE`**

enumerator **`LV_SCR_LOAD_ANIM_OVER_LEFT`**

enumerator **`LV_SCR_LOAD_ANIM_OVER_RIGHT`**

enumerator **`LV_SCR_LOAD_ANIM_OVER_TOP`**

enumerator **`LV_SCR_LOAD_ANIM_OVER_BOTTOM`**

enumerator **`LV_SCR_LOAD_ANIM_MOVE_LEFT`**

enumerator **`LV_SCR_LOAD_ANIM_MOVE_RIGHT`**

enumerator **`LV_SCR_LOAD_ANIM_MOVE_TOP`**

enumerator **`LV_SCR_LOAD_ANIM_MOVE_BOTTOM`**

enumerator **`LV_SCR_LOAD_ANIM_FADE_ON`**

Functions

lv_obj_t ***`lv_disp_get_scr_act`**(*lv_disp_t* *disp)
Return with a pointer to the active screen

Parameters **`disp`** -- pointer to display which active screen should be get. (NULL to use the default screen)

Returns pointer to the active screen object (loaded by 'lv_scr_load()')

lv_obj_t ***`lv_disp_get_scr_prev`**(*lv_disp_t* *disp)
Return with a pointer to the previous screen. Only used during screen transitions.

Parameters **`disp`** -- pointer to display which previous screen should be get. (NULL to use the default screen)

Returns pointer to the previous screen object or NULL if not used now

void **lv_disp_load_scr**(*lv_obj_t* *scr)

Make a screen active

Parameters **scr** -- pointer to a screen

lv_obj_t ***lv_disp_get_layer_top**(*lv_disp_t* *disp)

Return with the top layer. (Same on every screen and it is above the normal screen layer)

Parameters **disp** -- pointer to display which top layer should be get. (NULL to use the default screen)

Returns pointer to the top layer object (transparent screen sized lv_obj)

lv_obj_t ***lv_disp_get_layer_sys**(*lv_disp_t* *disp)

Return with the sys. layer. (Same on every screen and it is above the normal screen and the top layer)

Parameters **disp** -- pointer to display which sys. layer should be get. (NULL to use the default screen)

Returns pointer to the sys layer object (transparent screen sized lv_obj)

void **lv_disp_set_theme**(*lv_disp_t* *disp, *lv_theme_t* *th)

Get the theme of a display

Parameters **disp** -- pointer to a display

Returns the display's theme (can be NULL)

lv_theme_t ***lv_disp_get_theme**(*lv_disp_t* *disp)

Get the theme of a display

Parameters **disp** -- pointer to a display

Returns the display's theme (can be NULL)

void **lv_disp_set_bg_color**(*lv_disp_t* *disp, lv_color_t color)

Set the background color of a display

Parameters

- **disp** -- pointer to a display
- **color** -- color of the background

void **lv_disp_set_bg_image**(*lv_disp_t* *disp, const void *img_src)

Set the background image of a display

Parameters

- **disp** -- pointer to a display
- **img_src** -- path to file or pointer to an `lv_img_dsc_t` variable

void **lv_disp_set_bg_opa**(*lv_disp_t* *disp, lv_opa_t opa)

Opacity of the background

Parameters

- **disp** -- pointer to a display
- **opa** -- opacity (0..255)

void **lv_scr_load_anim**(*lv_obj_t* *scr, *lv_scr_load_anim_t* anim_type, uint32_t time, uint32_t delay, bool auto_del)

Switch screen with animation

Parameters

- **scr** -- pointer to the new screen to load
- **anim_type** -- type of the animation from `lv_scr_load_anim_t`. E.g. `LV_SCR_LOAD_ANIM_MOVE_LEFT`
- **time** -- time of the animation
- **delay** -- delay before the transition
- **auto_del** -- true: automatically delete the old screen

uint32_t **lv_disp_get_inactive_time**(const *lv_disp_t* *disp)

Get elapsed time since last user activity on a display (e.g. click)

Parameters **disp** -- pointer to an display (NULL to get the overall smallest inactivity)

Returns elapsed ticks (milliseconds) since the last activity

void **lv_disp_trig_activity**(*lv_disp_t* *disp)

Manually trigger an activity on a display

Parameters **disp** -- pointer to an display (NULL to use the default display)

void **lv_disp_clean_dcache**(*lv_disp_t* *disp)

Clean any CPU cache that is related to the display.

Parameters **disp** -- pointer to an display (NULL to use the default display)

lv_timer_t ***_lv_disp_get_refr_timer**(*lv_disp_t* *disp)

Get a pointer to the screen refresher timer to modify its parameters with `lv_timer_...` functions.

Parameters **disp** -- pointer to a display

Returns pointer to the display refresher timer. (NULL on error)

static inline *lv_obj_t* ***lv_scr_act**(void)

Get the active screen of the default display

Returns pointer to the active screen

static inline *lv_obj_t* ***lv_layer_top**(void)

Get the top layer of the default display

Returns pointer to the top layer

static inline *lv_obj_t* ***lv_layer_sys**(void)

Get the active screen of the default display

Returns pointer to the sys layer

static inline void **lv_scr_load**(*lv_obj_t* *scr)

static inline lv_coord_t **lv_dpx**(lv_coord_t n)

Scale the given number of pixels (a distance or size) relative to a 160 DPI display considering the DPI of the default display. It ensures that e.g. `lv_dpx(100)` will have the same physical size regardless to the DPI of the display.

Parameters **n** -- the number of pixels to scale

Returns `n x current_dpi/160`

static inline lv_coord_t **lv_disp_dpx**(const *lv_disp_t* *disp, lv_coord_t n)

Scale the given number of pixels (a distance or size) relative to a 160 DPI display considering the DPI of the given display. It ensures that e.g. `lv_dpx(100)` will have the same physical size regardless to the DPI of the display.

Parameters

- **obj** -- an display whose dpi should be considered
- **n** -- the number of pixels to scale

Returns `n x current_dpi/160`

Colors

Typedefs

typedef lv_color_t (*`lv_color_filter_cb_t`)(const struct *_lv_color_filter_dsc_t*, lv_color_t, lv_opa_t)

typedef struct *_lv_color_filter_dsc_t* `lv_color_filter_dsc_t`

Enums

enum `[anonymous]`

Opacity percentages.

Values:

enumerator `LV_OPA_TRANSP`

enumerator `LV_OPA_0`

enumerator `LV_OPA_10`

enumerator `LV_OPA_20`

enumerator `LV_OPA_30`

enumerator `LV_OPA_40`

enumerator `LV_OPA_50`

enumerator `LV_OPA_60`

enumerator `LV_OPA_70`

enumerator `LV_OPA_80`

enumerator `LV_OPA_90`

enumerator `LV_OPA_100`

enumerator `LV_OPA_COVER`

enum `lv_palette_t`

Values:

enumerator `LV_PALETTE_RED`

enumerator `LV_PALETTE_PINK`

enumerator `LV_PALETTE_PURPLE`

enumerator `LV_PALETTE_DEEP_PURPLE`

enumerator `LV_PALETTE_INDIGO`

enumerator `LV_PALETTE_BLUE`

enumerator **LV_PALETTE_LIGHT_BLUE**

enumerator **LV_PALETTE_CYAN**

enumerator **LV_PALETTE_TEAL**

enumerator **LV_PALETTE_GREEN**

enumerator **LV_PALETTE_LIGHT_GREEN**

enumerator **LV_PALETTE_LIME**

enumerator **LV_PALETTE_YELLOW**

enumerator **LV_PALETTE_AMBER**

enumerator **LV_PALETTE_ORANGE**

enumerator **LV_PALETTE_DEEP_ORANGE**

enumerator **LV_PALETTE_BROWN**

enumerator **LV_PALETTE_BLUE_GREY**

enumerator **LV_PALETTE_GREY**

enumerator **_LV_PALETTE_LAST**

enumerator **LV_PALETTE_NONE**

Functions

typedef LV_CONCAT3 (uint, LV_COLOR_SIZE, _t) lv_color_int_t

typedef LV_CONCAT3 (lv_color, LV_COLOR_DEPTH, _t) lv_color_t

static inline uint8_t **lv_color_to1**(lv_color_t color)

static inline uint8_t **lv_color_to8**(lv_color_t color)

static inline uint16_t **lv_color_to16**(lv_color_t color)

static inline uint32_t **lv_color_to32**(lv_color_t color)

static inline uint8_t **lv_color_brightness**(lv_color_t color)

Get the brightness of a color

Parameters **color** -- a color

Returns the brightness [0..255]

static inline lv_color_t **lv_color_make**(uint8_t r, uint8_t g, uint8_t b)

static inline lv_color_t **lv_color_hex**(uint32_t c)

static inline lv_color_t **lv_color_hex3**(uint32_t c)

static inline void **lv_color_filter_dsc_init**(*lv_color_filter_dsc_t* *dsc, *lv_color_filter_cb_t* cb)

lv_color_t **lv_color_lighten**(lv_color_t c, lv_opa_t lvl)

lv_color_t **lv_color_darken**(lv_color_t c, lv_opa_t lvl)

lv_color_t **lv_color_change_lightness**(lv_color_t c, lv_opa_t lvl)

lv_color_t **lv_color_hsv_to_rgb**(uint16_t h, uint8_t s, uint8_t v)

Convert a HSV color to RGB

Parameters

- **h** -- hue [0..359]
- **s** -- saturation [0..100]
- **v** -- value [0..100]

Returns the given RGB color in RGB (with LV_COLOR_DEPTH depth)

lv_color_hsv_t **lv_color_rgb_to_hsv**(uint8_t r8, uint8_t g8, uint8_t b8)

Convert a 32-bit RGB color to HSV

Parameters

- **r8** -- 8-bit red
- **g8** -- 8-bit green
- **b8** -- 8-bit blue

Returns the given RGB color in HSV

lv_color_hsv_t **lv_color_to_hsv**(lv_color_t color)

Convert a color to HSV

Parameters **color** -- color

Returns the given color in HSV

static inline lv_color_t **lv_color_chroma_key**(void)

Just a wrapper around LV_COLOR_CHROMA_KEY because it might be more convenient to use a function is some cases

Returns LV_COLOR_CHROMA_KEY

lv_color_t **lv_palette_main**(*lv_palette_t* p)

static inline lv_color_t **lv_color_white**(void)

static inline lv_color_t **lv_color_black**(void)

lv_color_t **lv_palette_lighten**(*lv_palette_t* p, uint8_t lvl)

lv_color_t **lv_palette_darken**(*lv_palette_t* p, uint8_t lvl)

union **lv_color1_t**

Public Members

uint8_t **full**

uint8_t **blue**

uint8_t **green**

uint8_t **red**

union *lv_color1_t*::[anonymous] **ch**

union **lv_color8_t**

Public Members

uint8_t **blue**

uint8_t **green**

uint8_t **red**

struct *lv_color8_t*::[anonymous] **ch**

uint8_t **full**

union **lv_color16_t**

Public Members

uint16_t **blue**

uint16_t **green**

uint16_t **red**

uint16_t **green_h**

uint16_t **green_l**

struct *lv_color16_t*::[anonymous] **ch**

uint16_t **full**

union **lv_color32_t**

Public Members

uint8_t **blue**

uint8_t **green**

uint8_t **red**

uint8_t **alpha**

struct *lv_color32_t*::[anonymous] **ch**

uint32_t **full**

struct **lv_color_hsv_t**

Public Members

uint16_t **h**

uint8_t **s**

uint8_t **v**

struct **_lv_color_filter_dsc_t**

Public Members

lv_color_filter_cb_t **filter_cb**

void ***user_data**

4.10 Fonts

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

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

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

The *bpp* also affects the required memory size to store the font. For example, *bpp = 4* makes the font nearly 4 times greater compared to *bpp = 1*.

4.10.1 Unicode support

LVGL supports **UTF-8** encoded Unicode characters. Your editor needs to be configureed to save your code/text as UTF-8 (usually this the default) and be sure that, `LV_TXT_ENC` is set to `LV_TXT_ENC_UTF8` in *lv_conf.h*. (This is the default value)

To test it try

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

If all works well, a ✓ character should be displayed.

4.10.2 Built-in fonts

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

Normal fonts

Containing all the ASCII characters, the degree symbol (U+00B0), the bullet symbol (U+2022) and the build in symbols (see below).

- `LV_FONT_MONTSERRAT_12` 12 px font
- `LV_FONT_MONTSERRAT_14` 14 px font
- `LV_FONT_MONTSERRAT_16` 16 px font
- `LV_FONT_MONTSERRAT_18` 18 px font
- `LV_FONT_MONTSERRAT_20` 20 px font
- `LV_FONT_MONTSERRAT_22` 22 px font
- `LV_FONT_MONTSERRAT_24` 24 px font
- `LV_FONT_MONTSERRAT_26` 26 px font
- `LV_FONT_MONTSERRAT_28` 28 px font
- `LV_FONT_MONTSERRAT_30` 30 px font
- `LV_FONT_MONTSERRAT_32` 32 px font
- `LV_FONT_MONTSERRAT_34` 34 px font
- `LV_FONT_MONTSERRAT_36` 36 px font
- `LV_FONT_MONTSERRAT_38` 38 px font
- `LV_FONT_MONTSERRAT_40` 40 px font
- `LV_FONT_MONTSERRAT_42` 42 px font
- `LV_FONT_MONTSERRAT_44` 44 px font
- `LV_FONT_MONTSERRAT_46` 46 px font
- `LV_FONT_MONTSERRAT_48` 48 px font

Special fonts

- `LV_FONT_MONTSERRAT_12_SUBPX` Same as normal 12 px font but with *subpixel rendering*
- `LV_FONT_MONTSERRAT_28_COMPRESSED` Same as normal 28 px font but *compressed font* with 3 bpp
- `LV_FONT_DEJAVU_16_PERSIAN_HEBREW` 16 px font with normal range + Hebrew, Arabic, Perisan letters and all their forms
- `LV_FONT_SIMSUN_16_CJK`16 px font with normal range + 1000 most common CJK radicals
- `LV_FONT_UNSCII_8` 8 px pixel perfect font with only ASCII characters
- `LV_FONT_UNSCII_16` 16 px pixel perfect font with only ASCII characters

The built-in fonts are **global variables** with names like `lv_font_montserrat_16` for 16 px hight font. To use them in a style, just add a pointer to a font variable like shown above.

The built-in fonts have *bpp = 4*, contains the ASCII characters and uses the Montserrat font.

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

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

The symbols can be used as:

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

Or with together with strings:

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

Or more symbols together:

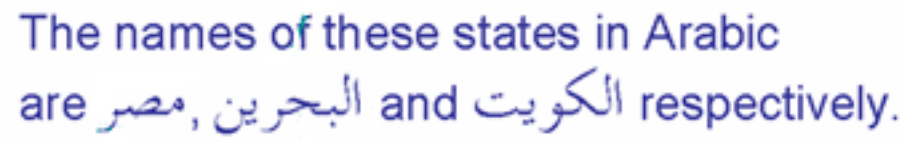
```
lv_label_set_text(my_label, LV_SYMBOL_OK LV_SYMBOL_WIFI LV_SYMBOL_PLAY);
```

4.10.3 Special features

Bidirectional support

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

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



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

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

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

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

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

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

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

Arabic and Persian support

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

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

However, there some limitations:

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

Subpixel rendering

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

Subpixel rendering requires to generate the fonts with special settings:

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

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

Compress fonts

The bitmaps of the fonts can be compressed by

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

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

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

4.10.4 Add new font

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

1. The simplest method is to use the Online font converter. Just set the parameters, click the *Convert* button, copy the font to your project and use it. **Be sure to carefully read the steps provided on that site or you will get an error while converting.**
2. Use the Offline font converter. (Requires Node.js to be installed)
3. If you want to create something like the built-in fonts (Roboto font and symbols) but in different size and/or ranges, you can use the `built_in_font_gen.py` script in `lvgl/scripts/built_in_font` folder. (It requires Python and `lv_font_conv` to be installed)

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

To make the fonts globally available (like the builtin fonts), add them to `LV_FONT_CUSTOM_DECLARE` in *lv_conf.h*.

4.10.5 Add new symbols

The built-in symbols are created from FontAwesome font.

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

Using the symbol

1. Convert the Unicode value to UTF8. You can do it e.g on this site. For `0xf287` the *Hex UTF-8 bytes* are `EF 8A 87`.
2. Create a `define` from the UTF8 values: `#define MY_USB_SYMBOL "\xEF\x8A\x87"`
3. Create a label and set the text. Eg. `lv_label_set_text(label, MY_USB_SYMBOL)`

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

4.10.6 Load font in run-time

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

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

Example

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

/*Use the font*/

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

4.10.7 Add a new font engine

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

A ready to use FreeType can be found in lv_freetype repository.

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

```
/*Describe the properties of a font*/
lv_font_t my_font;
my_font.get_glyph_dsc = my_get_glyph_dsc_cb;        /*Set a callback to get info␣
↪about gylphs*/
my_font.get_glyph_bitmap = my_get_glyph_bitmap_cb;  /*Set a callback to get bitmap of␣
↪a glyp*/
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*/

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

(continues on next page)

(continued from previous page)

```
}


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

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

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

4.11 Images

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

4.11.1 Store images

You can store images in two places

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

Variables

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

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

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

Files

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

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

4.11.2 Color formats

Various built-in color formats are supported:

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

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

For 32-bit color depth:

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

For 16-bit color depth:

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

For 8-bit color depth:

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

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

- **LV_IMG_CF_RAW** Indicates a basic raw image (e.g. a PNG or JPG image).
- **LV_IMG_CF_RAW_ALPHA** Indicates that the image has alpha and an alpha byte is added for every pixel.

- **LV_IMG_CF_RAW_CHROME_KEYED** Indicates that the image is chrome keyed as described in `LV_IMG_CF_TRUE_COLOR_CHROMA_KEYED` above.

4.11.3 Add and use images

You can add images to LVGL in two ways:

- using the online converter
- manually create images

Online converter

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

Adding an image to LVGL via online converter is easy.

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

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

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

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

Manually create an image

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

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

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

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

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

Use images

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

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

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

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

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

4.11.4 Image decoder

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

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

The image decoder consists of 4 callbacks:

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

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

The `LV_IMG_CF_TRUE_COLOR_...`, `LV_IMG_INDEXED_...` and `LV_IMG_ALPHA_...` formats (essentially, all non-`RAW` formats) are understood by the built-in decoder.

Custom image formats

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

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

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

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

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

Register an image decoder

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

First, you need to create a new image decoder and set some functions to open/close the PNG files. It should looks 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;
  header->w = width;
  header->h = height;
}

/**
 * Open a PNG image and return the decided 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;
```

(continues on next page)

(continued from previous page)

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

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

}
```

So in summary:

- In `decoder_info`, you should collect some basic information about the image and store it in `header`.
- In `decoder_open`, you should try to open the image source pointed by `dsc->src`. Its type is already in `dsc->src_type == LV_IMG_SRC_FILE/VARIABLE`. If this format/type is not supported by the decoder, return

`LV_RES_INV`. However, if you can open the image, a pointer to the decoded *True color* image should be set in `dsc->img_data`. If the format is known but, you don't want to decode while image (e.g. no memory for it) set `dsc->img_data = NULL` to call `read_line` to get the pixels.

- In `decoder_close` you should free all the allocated resources.
- `decoder_read` is optional. Decoding the whole image requires extra memory and some computational overhead. However, if can decode one line of the image without decoding the whole image, you can save memory and time. To indicate that, the *line read* function should be used, set `dsc->img_data = NULL` in the open function.

Manually use an image decoder

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

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

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

4.11.5 Image caching

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

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

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

Cache size

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

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

Value of images

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

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

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

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

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

Memory usage

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

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

Clean the cache

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

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

4.11.6 API

Image buffer

Typedefs

typedef uint8_t **`lv_img_cf_t`**

Enums

enum **[anonymous]**

Values:

enumerator **LV_IMG_CF_UNKNOWN**

enumerator **LV_IMG_CF_RAW**
Contains the file as it is. Needs custom decoder function

enumerator **LV_IMG_CF_RAW_ALPHA**
Contains the file as it is. The image has alpha. Needs custom decoder function

enumerator **LV_IMG_CF_RAW_CHROMA_KEYED**
Contains the file as it is. The image is chroma keyed. Needs custom decoder function

enumerator **LV_IMG_CF_TRUE_COLOR**
Color format and depth should match with LV_COLOR settings

enumerator **LV_IMG_CF_TRUE_COLOR_ALPHA**
Same as `LV_IMG_CF_TRUE_COLOR` but every pixel has an alpha byte

enumerator **LV_IMG_CF_TRUE_COLOR_CHROMA_KEYED**
Same as `LV_IMG_CF_TRUE_COLOR` but LV_COLOR_TRANSP pixels will be transparent

enumerator **LV_IMG_CF_INDEXED_1BIT**
Can have 2 different colors in a palette (always chroma keyed)

enumerator **LV_IMG_CF_INDEXED_2BIT**
Can have 4 different colors in a palette (always chroma keyed)

enumerator **LV_IMG_CF_INDEXED_4BIT**
Can have 16 different colors in a palette (always chroma keyed)

enumerator **LV_IMG_CF_INDEXED_8BIT**
Can have 256 different colors in a palette (always chroma keyed)

enumerator **LV_IMG_CF_ALPHA_1BIT**
Can have one color and it can be drawn or not

enumerator **LV_IMG_CF_ALPHA_2BIT**
Can have one color but 4 different alpha value

enumerator **LV_IMG_CF_ALPHA_4BIT**
Can have one color but 16 different alpha value

enumerator **LV_IMG_CF_ALPHA_8BIT**
Can have one color but 256 different alpha value

enumerator **LV_IMG_CF_RESERVED_15**
Reserved for further use.

enumerator **LV_IMG_CF_RESERVED_16**
Reserved for further use.

enumerator **LV_IMG_CF_RESERVED_17**
Reserved for further use.

enumerator **LV_IMG_CF_RESERVED_18**
Reserved for further use.

enumerator **LV_IMG_CF_RESERVED_19**
Reserved for further use.

enumerator **LV_IMG_CF_RESERVED_20**
Reserved for further use.

enumerator **LV_IMG_CF_RESERVED_21**
Reserved for further use.

enumerator **LV_IMG_CF_RESERVED_22**
Reserved for further use.

enumerator **LV_IMG_CF_RESERVED_23**
Reserved for further use.

enumerator **LV_IMG_CF_USER_ENCODED_0**
User holder encoding format.

enumerator **LV_IMG_CF_USER_ENCODED_1**
User holder encoding format.

enumerator **LV_IMG_CF_USER_ENCODED_2**
User holder encoding format.

enumerator **LV_IMG_CF_USER_ENCODED_3**
User holder encoding format.

enumerator **LV_IMG_CF_USER_ENCODED_4**
User holder encoding format.

enumerator **LV_IMG_CF_USER_ENCODED_5**
User holder encoding format.

enumerator **LV_IMG_CF_USER_ENCODED_6**
User holder encoding format.

enumerator **LV_IMG_CF_USER_ENCODED_7**
User holder encoding format.

Functions

lv_img_dsc_t ***lv_img_buf_alloc**(lv_coord_t w, lv_coord_t h, *lv_img_cf_t* cf)

Allocate an image buffer in RAM

Parameters

- **w** -- width of image
- **h** -- height of image
- **cf** -- a color format (`LV_IMG_CF_...`)

Returns an allocated image, or NULL on failure

lv_color_t **lv_img_buf_get_px_color**(*lv_img_dsc_t* *dsc, lv_coord_t x, lv_coord_t y, lv_color_t color)

Get the color of an image's pixel

Parameters

- **dsc** -- an image descriptor
- **x** -- x coordinate of the point to get
- **y** -- x coordinate of the point to get
- **color** -- the color of the image. In case of `LV_IMG_CF_ALPHA_1/2/4/8` this color is used. Not used in other cases.
- **safe** -- true: check out of bounds

Returns color of the point

lv_opa_t **lv_img_buf_get_px_alpha**(*lv_img_dsc_t* *dsc, lv_coord_t x, lv_coord_t y)

Get the alpha value of an image's pixel

Parameters

- **dsc** -- pointer to an image descriptor
- **x** -- x coordinate of the point to set
- **y** -- x coordinate of the point to set
- **safe** -- true: check out of bounds

Returns alpha value of the point

void **lv_img_buf_set_px_color**(*lv_img_dsc_t* *dsc, lv_coord_t x, lv_coord_t y, lv_color_t c)

Set the color of a pixel of an image. The alpha channel won't be affected.

Parameters

- **dsc** -- pointer to an image descriptor
- **x** -- x coordinate of the point to set
- **y** -- x coordinate of the point to set
- **c** -- color of the point
- **safe** -- true: check out of bounds

void **lv_img_buf_set_px_alpha**(*lv_img_dsc_t* *dsc, lv_coord_t x, lv_coord_t y, lv_opa_t opa)

Set the alpha value of a pixel of an image. The color won't be affected

Parameters

- **dsc** -- pointer to an image descriptor

- **x** -- x coordinate of the point to set
- **y** -- x coordinate of the point to set
- **opa** -- the desired opacity
- **safe** -- true: check out of bounds

void **lv_img_buf_set_palette**(*lv_img_dsc_t* *dsc, uint8_t id, lv_color_t c)

Set the palette color of an indexed image. Valid only for `LV_IMG_CF_INDEXED1/2/4/8`

Parameters

- **dsc** -- pointer to an image descriptor
- **id** -- the palette color to set:
  - for `LV_IMG_CF_INDEXED1`: 0..1
  - for `LV_IMG_CF_INDEXED2`: 0..3
  - for `LV_IMG_CF_INDEXED4`: 0..15
  - for `LV_IMG_CF_INDEXED8`: 0..255
- **c** -- the color to set

void **lv_img_buf_free**(*lv_img_dsc_t* *dsc)

Free an allocated image buffer

Parameters **dsc** -- image buffer to free

uint32_t **lv_img_buf_get_img_size**(lv_coord_t w, lv_coord_t h, *lv_img_cf_t* cf)

Get the memory consumption of a raw bitmap, given color format and dimensions.

Parameters

- **w** -- width
- **h** -- height
- **cf** -- color format

Returns size in bytes

void **_lv_img_buf_transform_init**(*lv_img_transform_dsc_t* *dsc)

Initialize a descriptor to rotate an image

Parameters **dsc** -- pointer to an *`lv_img_transform_dsc_t`* variable whose `cfg` field is initialized

bool **_lv_img_buf_transform_anti_alias**(*lv_img_transform_dsc_t* *dsc)

Continue transformation by taking the neighbors into account

Parameters **dsc** -- pointer to the transformation descriptor

bool **_lv_img_buf_transform**(*lv_img_transform_dsc_t* *dsc, lv_coord_t x, lv_coord_t y)

Get which color and opa would come to a pixel if it were rotated

Note: the result is written back to `dsc->res_color` and `dsc->res_opa`

Parameters

- **dsc** -- a descriptor initialized by `lv_img_buf_rotate_init`
- **x** -- the coordinate which color and opa should be get

- **y** -- the coordinate which color and opa should be get

Returns true: there is valid pixel on these x/y coordinates; false: the rotated pixel was out of the image

void **_lv_img_buf_get_transformed_area**(lv_area_t *res, lv_coord_t w, lv_coord_t h, int16_t angle, uint16_t zoom, const lv_point_t *pivot)

Get the area of a rectangle if its rotated and scaled

Parameters

- **res** -- store the coordinates here
- **w** -- width of the rectangle to transform
- **h** -- height of the rectangle to transform
- **angle** -- angle of rotation
- **zoom** -- zoom, (256 no zoom)
- **pivot** -- x,y pivot coordinates of rotation

struct **lv_img_header_t**

#include <lv_img_buf.h> The first 8 bit is very important to distinguish the different source types. For more info see `lv_img_get_src_type()` in lv_img.c On big endian systems the order is reversed so cf and always_zero must be at the end of the struct.

Public Members

uint32_t **h**

uint32_t **w**

uint32_t **reserved**

uint32_t **always_zero**

uint32_t **cf**

struct **lv_img_header_t**

#include <lv_img_buf.h> The first 8 bit is very important to distinguish the different source types. For more info see `lv_img_get_src_type()` in lv_img.c On big endian systems the order is reversed so cf and always_zero must be at the end of the struct.

Public Members

uint32_t **h**

uint32_t **w**

uint32_t **reserved**

uint32_t **always_zero**

uint32_t **cf**

struct **lv_img_dsc_t**

#include <lv_img_buf.h> Image header it is compatible with the result from image converter utility

Public Members

lv_img_header_t **header**
A header describing the basics of the image

uint32_t **data_size**
Size of the image in bytes

const uint8_t ***data**
Pointer to the data of the image

struct **lv_img_transform_dsc_t**

Public Members

const void ***src**

lv_coord_t **src_w**

lv_coord_t **src_h**

lv_coord_t **pivot_x**

lv_coord_t **pivot_y**

int16_t **angle**

uint16_t **zoom**

lv_color_t **color**

lv_img_cf_t **cf**

bool **antialias**

struct *lv_img_transform_dsc_t*::[anonymous] **cfg**

lv_opa_t **opa**

struct *lv_img_transform_dsc_t*::[anonymous] **res**

lv_img_dsc_t **img_dsc**

int32_t **pivot_x_256**

int32_t **pivot_y_256**

int32_t **sinma**

int32_t **cosma**

uint8_t **chroma_keyed**

uint8_t **has_alpha**

uint8_t **native_color**

uint32_t **zoom_inv**

lv_coord_t **xs**

lv_coord_t **ys**

lv_coord_t **xs_int**

lv_coord_t **ys_int**

uint32_t **pxi**

uint8_t **px_size**

struct *lv_img_transform_dsc_t*::[anonymous] **tmp**

4.12 File system

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

4.12.1 Add a driver

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

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

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


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

drv.free_space_cb = my_free_space_cb;     /*Callback to tell free space on the drive␣
↪*/

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

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

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

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

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

4.12.2 Usage example

The example below shows how to read from a file:

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

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

lv_fs_close(&f);
```

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

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

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

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

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

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

lv_fs_dir_close(&dir);
```

4.12.3 Use drivers for images

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

To initialize the image, the following callbacks are required:

- open
- close
- read
- seek
- tell

4.12.4 API

Typedefs

typedef uint8_t **`lv_fs_res_t`**

typedef uint8_t **`lv_fs_mode_t`**

typedef uint8_t **`lv_fs_whence_t`**

typedef struct *_lv_fs_drv_t* **`lv_fs_drv_t`**

Enums

enum **`[anonymous]`**

Errors in the file system module.

Values:

enumerator **`LV_FS_RES_OK`**

enumerator **`LV_FS_RES_HW_ERR`**

enumerator **`LV_FS_RES_FS_ERR`**

enumerator **`LV_FS_RES_NOT_EX`**

enumerator **`LV_FS_RES_FULL`**

enumerator **`LV_FS_RES_LOCKED`**

enumerator **`LV_FS_RES_DENIED`**

enumerator **`LV_FS_RES_BUSY`**

enumerator **`LV_FS_RES_TOUT`**

enumerator **`LV_FS_RES_NOT_IMP`**

enumerator **`LV_FS_RES_OUT_OF_MEM`**

enumerator **`LV_FS_RES_INV_PARAM`**

enumerator **`LV_FS_RES_UNKNOWN`**

enum **[anonymous]**
File open mode.

Values:

enumerator **LV_FS_MODE_WR**

enumerator **LV_FS_MODE_RD**

enum **[anonymous]**
Seek modes.

Values:

enumerator **LV_FS_SEEK_SET**

enumerator **LV_FS_SEEK_CUR**

enumerator **LV_FS_SEEK_END**

Functions

void **_lv_fs_init**(void)
Initialize the File system interface

void **lv_fs_drv_init**(*lv_fs_drv_t* *drv)
Initialize a file system driver with default values. It is used to surly have known values in the fields ant not memory junk. After it you can set the fields.

Parameters **drv** -- pointer to driver variable to initialize

void **lv_fs_drv_register**(*lv_fs_drv_t* *drv_p)
Add a new drive

Parameters **drv_p** -- pointer to an lv_fs_drv_t structure which is inited with the corresponding function pointers. Only pointer is saved, so the driver should be static or dynamically allocated.

lv_fs_drv_t ***lv_fs_get_drv**(char letter)
Give a pointer to a driver from its letter

Parameters **letter** -- the driver letter

Returns pointer to a driver or NULL if not found

bool **lv_fs_is_ready**(char letter)
Test if a drive is ready or not. If the `ready` function was not initialized `true` will be returned.

Parameters **letter** -- letter of the drive

Returns true: drive is ready; false: drive is not ready

lv_fs_res_t **lv_fs_open**(*lv_fs_file_t* *file_p, const char *path, *lv_fs_mode_t* mode)
Open a file

Parameters

- **file_p** -- pointer to a *lv_fs_file_t* variable
- **path** -- path to the file beginning with the driver letter (e.g. S:/folder/file.txt)
- **mode** -- read: FS_MODE_RD, write: FS_MODE_WR, both: FS_MODE_RD | FS_MODE_WR

Returns LV_FS_RES_OK or any error from lv_fs_res_t enum

lv_fs_res_t **lv_fs_close**(*lv_fs_file_t* *file_p)

Close an already opened file

Parameters **file_p** -- pointer to a *lv_fs_file_t* variable

Returns LV_FS_RES_OK or any error from lv_fs_res_t enum

lv_fs_res_t **lv_fs_read**(*lv_fs_file_t* *file_p, void *buf, uint32_t btr, uint32_t *br)

Read from a file

Parameters

- **file_p** -- pointer to a *lv_fs_file_t* variable
- **buf** -- pointer to a buffer where the read bytes are stored
- **btr** -- Bytes To Read
- **br** -- the number of real read bytes (Bytes Read). NULL if unused.

Returns LV_FS_RES_OK or any error from lv_fs_res_t enum

lv_fs_res_t **lv_fs_write**(*lv_fs_file_t* *file_p, const void *buf, uint32_t btw, uint32_t *bw)

Write into a file

Parameters

- **file_p** -- pointer to a *lv_fs_file_t* variable
- **buf** -- pointer to a buffer with the bytes to write
- **btr** -- Bytes To Write
- **br** -- the number of real written bytes (Bytes Written). NULL if unused.

Returns LV_FS_RES_OK or any error from lv_fs_res_t enum

lv_fs_res_t **lv_fs_seek**(*lv_fs_file_t* *file_p, uint32_t pos, *lv_fs_whence_t* whence)

Set the position of the 'cursor' (read write pointer) in a file

Parameters

- **file_p** -- pointer to a *lv_fs_file_t* variable
- **pos** -- the new position expressed in bytes index (0: start of file)

Returns LV_FS_RES_OK or any error from lv_fs_res_t enum

lv_fs_res_t **lv_fs_tell**(*lv_fs_file_t* *file_p, uint32_t *pos)

Give the position of the read write pointer

Parameters

- **file_p** -- pointer to a *lv_fs_file_t* variable
- **pos_p** -- pointer to store the position of the read write pointer

Returns LV_FS_RES_OK or any error from 'fs_res_t'

lv_fs_res_t **lv_fs_dir_open**(*lv_fs_dir_t* *rddir_p, const char *path)

Initialize a 'fs_dir_t' variable for directory reading

Parameters

- **rddir_p** -- pointer to a '*lv_fs_dir_t*' variable
- **path** -- path to a directory

Returns LV_FS_RES_OK or any error from lv_fs_res_t enum

lv_fs_res_t **lv_fs_dir_read**(*lv_fs_dir_t* *rddir_p, char *fn)

Read the next filename form a directory. The name of the directories will begin with '/'

Parameters

- **rddir_p** -- pointer to an initialized 'fs_dir_t' variable
- **fn** -- pointer to a buffer to store the filename

Returns LV_FS_RES_OK or any error from lv_fs_res_t enum

lv_fs_res_t **lv_fs_dir_close**(*lv_fs_dir_t* *rddir_p)

Close the directory reading

Parameters **rddir_p** -- pointer to an initialized 'fs_dir_t' variable

Returns LV_FS_RES_OK or any error from lv_fs_res_t enum

char ***lv_fs_get_letters**(char *buf)

Fill a buffer with the letters of existing drivers

Parameters **buf** -- buffer to store the letters ('\0' added after the last letter)

Returns the buffer

const char ***lv_fs_get_ext**(const char *fn)

Return with the extension of the filename

Parameters **fn** -- string with a filename

Returns pointer to the beginning extension or empty string if no extension

char ***lv_fs_up**(char *path)

Step up one level

Parameters **path** -- pointer to a file name

Returns the truncated file name

const char ***lv_fs_get_last**(const char *path)

Get the last element of a path (e.g. U:/folder/file -> file)

Parameters **path** -- pointer to a file name

Returns pointer to the beginning of the last element in the path

struct **_lv_fs_drv_t**

Public Members

char **letter**

bool (***ready_cb**)(struct *_lv_fs_drv_t* *drv)

void *(***open_cb**)(struct *_lv_fs_drv_t* *drv, const char *path, *lv_fs_mode_t* mode)

lv_fs_res_t (***close_cb**)(struct *_lv_fs_drv_t* *drv, void *file_p)

lv_fs_res_t (***read_cb**)(struct *_lv_fs_drv_t* *drv, void *file_p, void *buf, uint32_t btr, uint32_t *br)

lv_fs_res_t (***write_cb**)(struct *_lv_fs_drv_t* *drv, void *file_p, const void *buf, uint32_t btw, uint32_t *bw)

lv_fs_res_t (***seek_cb**)(struct *_lv_fs_drv_t* *drv, void *file_p, uint32_t pos, *lv_fs_whence_t* whence)

lv_fs_res_t (*`tell_cb`)(struct *_lv_fs_drv_t* *drv, void *file_p, uint32_t *pos_p)

void *(*`dir_open_cb`)(struct *_lv_fs_drv_t* *drv, const char *path)

lv_fs_res_t (*`dir_read_cb`)(struct *_lv_fs_drv_t* *drv, void *rddir_p, char *fn)

lv_fs_res_t (*`dir_close_cb`)(struct *_lv_fs_drv_t* *drv, void *rddir_p)

void *`user_data`
Custom file user data

struct `lv_fs_file_t`

Public Members

void *`file_d`

lv_fs_drv_t *`drv`

struct `lv_fs_dir_t`

Public Members

void *`dir_d`

lv_fs_drv_t *`drv`

4.13 Animations

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

The *animator* functions has the following prototype:

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

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

4.13.1 Create an animation

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

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

lv_anim_t a;
lv_anim_init(&a);

/* MANDATORY SETTINGS
```

(continues on next page)

(continued from previous page)

```
 *------------------*/

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

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

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

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

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

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

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

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

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

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

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

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

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

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

/* START THE ANIMATION
 *------------------*/
lv_anim_start(&a);                              /*Start the animation*/
```

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

4.13.2 Animation path

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

- **lv_anim_path_linear** linear animation
- **lv_anim_path_step** change in one step at the end
- **lv_anim_path_ease_in** slow at the beginning
- **lv_anim_path_ease_out** slow at the end
- **lv_anim_path_ease_in_out** slow at the beginning and end too
- **lv_anim_path_overshoot** overshoot the end value
- **lv_anim_path_bounce** bounce back a little from the end value (like hitting a wall)

A path can be initialized like this:

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

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

4.13.3 Speed vs time

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

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

4.13.4 Delete animations

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

4.13.5 API

Input device

Typedefs

typedef int32_t (***`lv_anim_path_cb_t`**)(const struct *_lv_anim_t**)
Get the current value during an animation

typedef void (*`lv_anim_exec_xcb_t`)(void*, int32_t)
Generic prototype of "animator" functions. First parameter is the variable to animate. Second parameter is the value to set. Compatible with `lv_xxx_set_yyy(obj, value)` functions The `x` in `_xcb_t` means its not a fully generic prototype because it doesn't receive `lv_anim_t *` as its first argument

typedef void (*`lv_anim_custom_exec_cb_t`)(struct *_lv_anim_t*, int32_t)
Same as `lv_anim_exec_xcb_t` but receives `lv_anim_t *` as the first parameter. It's more consistent but less convenient. Might be used by binding generator functions.

typedef void (*`lv_anim_ready_cb_t`)(struct *_lv_anim_t*)
Callback to call when the animation is ready

typedef void (*`lv_anim_start_cb_t`)(struct *_lv_anim_t*)
Callback to call when the animation really stars (considering `delay`)

typedef int32_t (*`lv_anim_get_value_cb_t`)(struct *_lv_anim_t*)
Callback used when the animation values are relative to get the current value

typedef struct *_lv_anim_t* `lv_anim_t`
Describes an animation

Enums

enum `lv_anim_enable_t`
Can be used to indicate if animations are enabled or disabled in a case

Values:

enumerator `LV_ANIM_OFF`

enumerator `LV_ANIM_ON`

Functions

`LV_EXPORT_CONST_INT`(LV_ANIM_REPEAT_INFINITE)

void `_lv_anim_core_init`(void)
Init. the animation module

void `lv_anim_init`(*lv_anim_t* *a)
Initialize an animation variable. E.g.: lv_anim_t a; lv_anim_init(&a); lv_anim_set_...(&a);

Parameters `a` -- pointer to an `lv_anim_t` variable to initialize

static inline void `lv_anim_set_var`(*lv_anim_t* *a, void *var)
Set a variable to animate

Parameters

- `a` -- pointer to an initialized `lv_anim_t` variable
- `var` -- pointer to a variable to animate

static inline void **lv_anim_set_exec_cb**(*lv_anim_t* *a, *lv_anim_exec_xcb_t* exec_cb)
Set a function to animate `var`

Parameters

- **a** -- pointer to an initialized `lv_anim_t` variable
- **exec_cb** -- a function to execute during animation LittelvGL's built-in functions can be used. E.g. lv_obj_set_x

static inline void **lv_anim_set_time**(*lv_anim_t* *a, uint32_t duration)
Set the duration of an animation

Parameters

- **a** -- pointer to an initialized `lv_anim_t` variable
- **duration** -- duration of the animation in milliseconds

static inline void **lv_anim_set_delay**(*lv_anim_t* *a, uint32_t delay)
Set a delay before starting the animation

Parameters

- **a** -- pointer to an initialized `lv_anim_t` variable
- **delay** -- delay before the animation in milliseconds

static inline void **lv_anim_set_values**(*lv_anim_t* *a, int32_t start, int32_t end)
Set the start and end values of an animation

Parameters

- **a** -- pointer to an initialized `lv_anim_t` variable
- **start** -- the start value
- **end** -- the end value

static inline void **lv_anim_set_custom_exec_cb**(*lv_anim_t* *a, *lv_anim_custom_exec_cb_t* exec_cb)
Similar to `lv_anim_set_exec_cb` but `lv_anim_custom_exec_cb_t` receives `lv_anim_t *` as its first parameter instead of `void *`. This function might be used when LVGL is binded to other languages because it's more consistent to have `lv_anim_t *` as first parameter. The variable to animate can be stored in the animation's `user_sata`

Parameters

- **a** -- pointer to an initialized `lv_anim_t` variable
- **exec_cb** -- a function to execute.

static inline void **lv_anim_set_path_cb**(*lv_anim_t* *a, *lv_anim_path_cb_t* path_cb)
Set the path (curve) of the animation.

Parameters

- **a** -- pointer to an initialized `lv_anim_t` variable
- **path_cb** -- a function the get the current value of the animation.

static inline void **lv_anim_set_start_cb**(*lv_anim_t* *a, *lv_anim_ready_cb_t* start_cb)
Set a function call when the animation really starts (considering `delay`)

Parameters

- **a** -- pointer to an initialized `lv_anim_t` variable
- **start_cb** -- a function call when the animation starts

static inline void **lv_anim_set_get_value_cb**(*lv_anim_t* *a, *lv_anim_get_value_cb_t* get_value_cb)
Set a function to use the current value of the variable and make start and end value relative the the returned current value.

Parameters

- **a** -- pointer to an initialized `lv_anim_t` variable
- **get_value_cb** -- a function call when the animation starts

static inline void **lv_anim_set_ready_cb**(*lv_anim_t* *a, *lv_anim_ready_cb_t* ready_cb)
Set a function call when the animation is ready

Parameters

- **a** -- pointer to an initialized `lv_anim_t` variable
- **ready_cb** -- a function call when the animation is ready

static inline void **lv_anim_set_playback_time**(*lv_anim_t* *a, uint32_t time)
Make the animation to play back to when the forward direction is ready

Parameters

- **a** -- pointer to an initialized `lv_anim_t` variable
- **time** -- the duration of the playback animation in in milliseconds. 0: disable playback

static inline void **lv_anim_set_playback_delay**(*lv_anim_t* *a, uint32_t delay)
Make the animation to play back to when the forward direction is ready

Parameters

- **a** -- pointer to an initialized `lv_anim_t` variable
- **delay** -- delay in milliseconds before starting the playback animation.

static inline void **lv_anim_set_repeat_count**(*lv_anim_t* *a, uint16_t cnt)
Make the animation repeat itself.

Parameters

- **a** -- pointer to an initialized `lv_anim_t` variable
- **cnt** -- repeat count or `LV_ANIM_REPEAT_INFINITE` for infinite repetition. 0: to disable repetition.

static inline void **lv_anim_set_repeat_delay**(*lv_anim_t* *a, uint32_t delay)
Set a delay before repeating the animation.

Parameters

- **a** -- pointer to an initialized `lv_anim_t` variable
- **delay** -- delay in milliseconds before repeating the animation.

static inline void **lv_anim_set_early_apply**(*lv_anim_t* *a, bool en)
Set a whether the animation's should be applied immediately or only when the delay expired.

Parameters

- **a** -- pointer to an initialized `lv_anim_t` variable
- **en** -- true: apply the start value immediately in `lv_anim_start`; false: apply the start value only when `delay` ms is elapsed and the animations really starts

void **lv_anim_start**(*lv_anim_t* *a)
Create an animation

Parameters **a** -- an initialized 'anim_t' variable. Not required after call.

static inline uint32_t **lv_anim_get_delay**(*lv_anim_t* *a)

Get a delay before starting the animation

Parameters **a** -- pointer to an initialized `lv_anim_t` variable

Returns delay before the animation in milliseconds

bool **lv_anim_del**(void *var, *lv_anim_exec_xcb_t* exec_cb)

Delete an animation of a variable with a given animator function

Parameters

- **var** -- pointer to variable
- **exec_cb** -- a function pointer which is animating 'var', or NULL to ignore it and delete all the animations of 'var

Returns true: at least 1 animation is deleted, false: no animation is deleted

void **lv_anim_del_all**(void)

Delete all the animations animation

lv_anim_t ***lv_anim_get**(void *var, *lv_anim_exec_xcb_t* exec_cb)

Get the animation of a variable and its `exec_cb`.

Parameters

- **var** -- pointer to variable
- **exec_cb** -- a function pointer which is animating 'var', or NULL to delete all the animations of 'var'

Returns pointer to the animation.

static inline bool **lv_anim_custom_del**(*lv_anim_t* *a, *lv_anim_custom_exec_cb_t* exec_cb)

Delete an animation by getting the animated variable from `a`. Only animations with `exec_cb` will be deleted. This function exists because it's logical that all anim. functions receives an `lv_anim_t` as their first parameter. It's not practical in C but might make the API more consequent and makes easier to generate bindings.

Parameters

- **a** -- pointer to an animation.
- **exec_cb** -- a function pointer which is animating 'var', or NULL to ignore it and delete all the animations of 'var

Returns true: at least 1 animation is deleted, false: no animation is deleted

uint16_t **lv_anim_count_running**(void)

Get the number of currently running animations

Returns the number of running animations

uint32_t **lv_anim_speed_to_time**(uint32_t speed, int32_t start, int32_t end)

Calculate the time of an animation with a given speed and the start and end values

Parameters

- **speed** -- speed of animation in unit/sec
- **start** -- start value of the animation
- **end** -- end value of the animation

Returns the required time [ms] for the animation with the given parameters

void **lv_anim_refr_now**(void)

Manually refresh the state of the animations. Useful to make the animations running in a blocking process where `lv_timer_handler` can't run for a while. Shouldn't be used directly because it is called in `lv_refr_now()`.

int32_t **lv_anim_path_linear**(const *lv_anim_t* *a)

Calculate the current value of an animation applying linear characteristic

Parameters **a** -- pointer to an animation

Returns the current value to set

int32_t **lv_anim_path_ease_in**(const *lv_anim_t* *a)

Calculate the current value of an animation slowing down the start phase

Parameters **a** -- pointer to an animation

Returns the current value to set

int32_t **lv_anim_path_ease_out**(const *lv_anim_t* *a)

Calculate the current value of an animation slowing down the end phase

Parameters **a** -- pointer to an animation

Returns the current value to set

int32_t **lv_anim_path_ease_in_out**(const *lv_anim_t* *a)

Calculate the current value of an animation applying an "S" characteristic (cosine)

Parameters **a** -- pointer to an animation

Returns the current value to set

int32_t **lv_anim_path_overshoot**(const *lv_anim_t* *a)

Calculate the current value of an animation with overshoot at the end

Parameters **a** -- pointer to an animation

Returns the current value to set

int32_t **lv_anim_path_bounce**(const *lv_anim_t* *a)

Calculate the current value of an animation with 3 bounces

Parameters **a** -- pointer to an animation

Returns the current value to set

int32_t **lv_anim_path_step**(const *lv_anim_t* *a)

Calculate the current value of an animation applying step characteristic. (Set end value on the end of the animation)

Parameters **a** -- pointer to an animation

Returns the current value to set

struct **_lv_anim_t**

#include <lv_anim.h> Describes an animation

Public Members

void *`var`
Variable to animate

lv_anim_exec_xcb_t `exec_cb`
Function to execute to animate

lv_anim_start_cb_t `start_cb`
Call it when the animation is starts (considering `delay`)

lv_anim_ready_cb_t `ready_cb`
Call it when the animation is ready

lv_anim_get_value_cb_t `get_value_cb`
Get the current value in relative mode

void *`user_data`
Custom user data

lv_anim_path_cb_t `path_cb`
Describe the path (curve) of animations

int32_t `start_value`
Start value

int32_t `current_value`
Current value

int32_t `end_value`
End value

int32_t `time`
Animation time in ms

int32_t `act_time`
Current time in animation. Set to negative to make delay.

uint32_t `playback_delay`
Wait before play back

uint32_t `playback_time`
Duration of playback animation

uint32_t `repeat_delay`
Wait before repeat

uint16_t `repeat_cnt`
Repeat count for the animation

uint8_t **early_apply**
1: Apply start value immediately even is there is `delay`

uint8_t **playback_now**
Play back is in progress

uint8_t **run_round**
Indicates the animation has run in this round

uint8_t **start_cb_called**
Indicates that the `start_cb` was already called

uint32_t **time_orig**

4.14 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 periodically every few milliseconds. See Porting for more information.

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

4.14.1 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. It allows you to create a new timer without specifying any parameters.

A timer callback should have `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);
```

4.14.2 Ready and Reset

`lv_timer_ready(timer)` makes the 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.

4.14.3 Set parameters

You can modify some parameters of the timers later:

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

4.14.4 Repeat count

You can make a timer repat only a given times with `lv_timer_set_repeat_count(timer, count)`. The timer will automatically be deleted after being called the defined times. Set the ount to `-1` to repeat infinitly.

4.14.5 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 an timer, as it won't actually measure the time the OS spends in an idle thread.

4.14.6 Asynchronous calls

In some cases, you can't do 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 make `my_function` be called on the next call of `lv_timer_handler`. `data_p` will be passed to function when it's called. Note that, only the pointer of the data is saved so you need to ensure that the variable will be "alive" while the function is called. It can be *static*, global or dynamically allocated data.

For example:

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

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

...

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

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

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

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

4.14.7 API

Typedefs

typedef void (*`lv_timer_cb_t`)(struct *_lv_timer_t*)
Timers execute this type of functions.

typedef struct *_lv_timer_t* `lv_timer_t`
Descriptor of a lv_timer

Functions

void `_lv_timer_core_init`(void)
Init the lv_timer module

lv_timer_t *`lv_timer_create_basic`(void)
Create an "empty" timer. It needs to initialized with at least `lv_timer_set_cb` and `lv_timer_set_period`

Returns pointer to the created timer

lv_timer_t *`lv_timer_create`(*lv_timer_cb_t* timer_xcb, uint32_t period, void *user_data)
Create a new lv_timer

Parameters

- **`timer_xcb`** -- a callback to call periodically. (the 'x' in the argument name indicates that its not a fully generic function because it not follows the `func_name(object, callback, ...)` convention)
- **`period`** -- call period in ms unit
- **`user_data`** -- custom parameter

Returns pointer to the new timer

void `lv_timer_del`(*lv_timer_t* *timer)
Delete a lv_timer

Parameters **`timer`** -- pointer to an lv_timer

void `lv_timer_pause`(*lv_timer_t* *timer)
Pause/resume a timer.

Parameters

- **`timer`** -- pointer to an lv_timer
- **`pause`** -- true: pause the timer; false: resume

void `lv_timer_resume`(*lv_timer_t* *timer)

void `lv_timer_set_cb`(*lv_timer_t* *timer, *lv_timer_cb_t* timer_cb)
Set the callback the timer (the function to call periodically)

Parameters

- **timer** -- pointer to a timer
- **timer_cb** -- the function to call periodically

void **lv_timer_set_period**(*lv_timer_t* *timer, uint32_t period)
Set new period for a lv_timer

Parameters

- **timer** -- pointer to a lv_timer
- **period** -- the new period

void **lv_timer_ready**(*lv_timer_t* *timer)
Make a lv_timer ready. It will not wait its period.

Parameters **timer** -- pointer to a lv_timer.

void **lv_timer_set_repeat_count**(*lv_timer_t* *timer, int32_t repeat_count)
Set the number of times a timer will repeat.

Parameters

- **timer** -- pointer to a lv_timer.
- **repeat_count** -- -1 : infinity; 0 : stop ; n>0: residual times

void **lv_timer_reset**(*lv_timer_t* *timer)
Reset a lv_timer. It will be called the previously set period milliseconds later.

Parameters **timer** -- pointer to a lv_timer.

void **lv_timer_enable**(bool en)
Enable or disable the whole lv_timer handling

Parameters **en** -- true: lv_timer handling is running, false: lv_timer handling is suspended

uint8_t **lv_timer_get_idle**(void)
Get idle percentage

Returns the lv_timer idle in percentage

lv_timer_t ***lv_timer_get_next**(*lv_timer_t* *timer)
Iterate through the timers

Parameters **timer** -- NULL to start iteration or the previous return value to get the next timer

Returns the next timer or NULL if there is no more timer

struct **_lv_timer_t**
#include <lv_timer.h> Descriptor of a lv_timer

Public Members

uint32_t **period**
How often the timer should run

uint32_t **last_run**
Last time the timer ran

lv_timer_cb_t **timer_cb**
Timer function

void ***user_data**
Custom user data

int32_t **repeat_count**
1: One time; -1 : infinity; n>0: residual times

uint32_t **paused**

Typedefs

typedef void (***lv_async_cb_t**)(void*)
Type for async callback.

Functions

lv_res_t **lv_async_call**(*lv_async_cb_t* async_xcb, void *user_data)
Call an asynchronous function the next time lv_timer_handler() is run. This function is likely to return **before** the call actually happens!

Parameters

- **async_xcb** -- a callback which is the task itself. (the 'x' in the argument name indicates that its not a fully generic function because it not follows the `func_name(object, callback, ...)` convention)
- **user_data** -- custom parameter

4.15 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 might 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 to the screen, but draw to an internal draw buffer first. When drawing (rendering) is ready copy that buffer to the screen.

The draw buffer can be smaller than the screen'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 screen:

1. It avoids flickering while the layers of the UI are drawn. For example, if LVGL drawn directly into 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 2 screen 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 2 frame buffers (which usually requires external RAM) but only smaller draw buffer(s) that can easily fit into the internal RAM too.

4.15.1 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 on the UI which requires redrawing. For example, a button is pressed, a chart is changed or an animation happened, etc.
2. LVGL saves the changed object's old and new area into a buffer, called an *Invalid area buffer*. For optimization, in some cases, objects are not added to the buffer:
   - Hidden objects are not added.
   - Objects completely out of their parent are not added.
   - Areas partially out of the parent are cropped to the parent's area.
   - The objects on other screens are not added.
3. In every `LV_DISP_DEF_REFR_PERIOD` (set in `lv_conf.h`) the followings happen:
   - LVGL checks the invalid areas and joins the adjacent or intersecting areas.
   - Takes the first joined area, if it's smaller than the *draw buffer*, then simply render 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.
   - Do the same with all the 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 required to draw the screen under 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 address of the frame buffer.

4.15.2 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, you might find interesting to know how drawing works under hood. Knowing about mask comes in handy if you want to customize drawing.

To learn masking let's learn 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`). It tells the parameters of drawing, for example the 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 renders 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. Else create the required masks (e.g. a rounded rectangle mask)

4. **Calculate all the added mask**. It creates 0..255 values into a *mask buffer* with the "shape" of the created masks. E.g. in case of a "line mask" according to the parameters of the mask, keep one side of the buffer as it is (255 by default) and set the rest to 0 to indicate that this side should be removed.
5. **Blend a color or image** During blending masks (make some pixels transparent or opaque), blending modes (additive, subtractive, etc), 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 4 of it. Essentially, every (skew) line is bounded with 4 line masks by forming a rectangle.
- `LV_DRAW_MASK_TYPE_RADIUS` Removes the inner or outer parts of a rectangle which can have radius. It's also used to create circles by setting the radius to large value (`LV_RADIUS_CIRCLE`)
- `LV_DRAW_MASK_TYPE_ANGLE` Removes a circle sector. It is used by `lv_draw_arc` to remove the "empty" sector.
- `LV_DRAW_MASK_TYPE_FADE` Create a vertical fade (change opacity)
- `LV_DRAW_MASK_TYPE_MAP` The mask is stored in an array and the necessary parts are applied

Masks are used the create almost every basic primitives:

- **letters** Create a mask from the letter and draw a rectangle with the letter's color considering the mask.
- **line** Created from 4 "line masks", to mask out the left, right, top and bottom part of the line to get perfectly perpendicular line ending.
- **rounded rectangle** A mask is created real-time to add radius to the corners.
- **clip corner** To clip to overflowing content (usually children) on the rounded corners also a rounded rectangle mask is applied.
- **rectangle border** Same as a rounded rectangle, but inner part is masked out too.
- **arc drawing** A circle border is drawn, but an arc mask is applied too.
- **ARGB images** The alpha channel is separated into a mask and the image is drawn as a normal RGB image.

4.15.3 Hook drawing

Although widgets can be very well customized by styles there might be cases when something really custom is required. To ensure a great level of flexibility LVGL sends a lot events during drawing with parameters that tells what LVGL is about to draw. Some fields of these parameters can be modified to draw something else or any custom drawing can be added manually.

A good use case for it 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 ever button and you can tell LVGL for example to use different colors on a specific buttons or manually draw an image on an some buttons.

Below each related events are described in detail.

Main drawing

These events are related to the actual drawing of the object. E.g. 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 limited area.

LV_EVENT_DRAW_MAIN_BEGIN

Sent before starting to draw an object. It's 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 the 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 good place to remove the 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 the scrollbars because they should be above all 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 happens here.

LV_EVENT_DRAW_POST_END

Called when post drawing has finished. If the 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. It allows changing the parts on a very low level with masks, extra drawing, or changing the parameters the 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 see 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 desciptors, 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 paramters
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 chart or the center of an arc.
const lv_point_t * p2;              // A point calculated during drawing. E.g. a␣
↪point of 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. It's 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. It's a good place to draw extra content on the part, or remove the 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 an 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 areas is fully covered by the object
- `LV_COVER_RES_NOT_COVER` the areas is not covered by the object
- `LV_COVER_RES_MASKED` there is a mask on the object so it can not covert the area

Here are some cases why can't an object fully cover an area:

- It's simply not fully on the that area
- It has radius
- It has not 100% background opacity
- It's an ARGB or chroma keyed image
- It has not normal blending mode. In this case LVGL needs to know the colors under the object to make the blending properly
- It's a text, etc

In short if for any reason the the area below the object is visible than it 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 widget are handled in the widget's 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 widget can't know about it.

LV_EVENT_REFR_EXT_DRAW_SIZE

If you need to draw outside of a widget LVGL needs to know about it to provide the extra space for drawing. Let's say you create an event the 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 simple set the required draw area with `lv_event_set_ext_draw_size(e, size)`.

4.16 New widget

CHAPTER

FIVE

WIDGETS

5.1 Base object (lv_obj)

5.1.1 Overview

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

- coordinates
- parent object
- children
- 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 Base object can be used with other widgets too. For example `lv_obj_set_width(slider, 100)`

The Base object can be directly used as a simple widgets. It nothing else than a rectangle. Or from the the HTML world it's like a `<div>`.

Coordinates

Here only a small subset of cooridnate settings is described. To see all the features of LVGL (padding, cooridnates 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 x and y coordinates relative to the parent with `lv_obj_set_x(obj, new_x)` and `lv_obj_set_y(obj, new_y)`, or both at the same time with `lv_obj_set_pos(obj, new_x, new_y)`.

Alignment

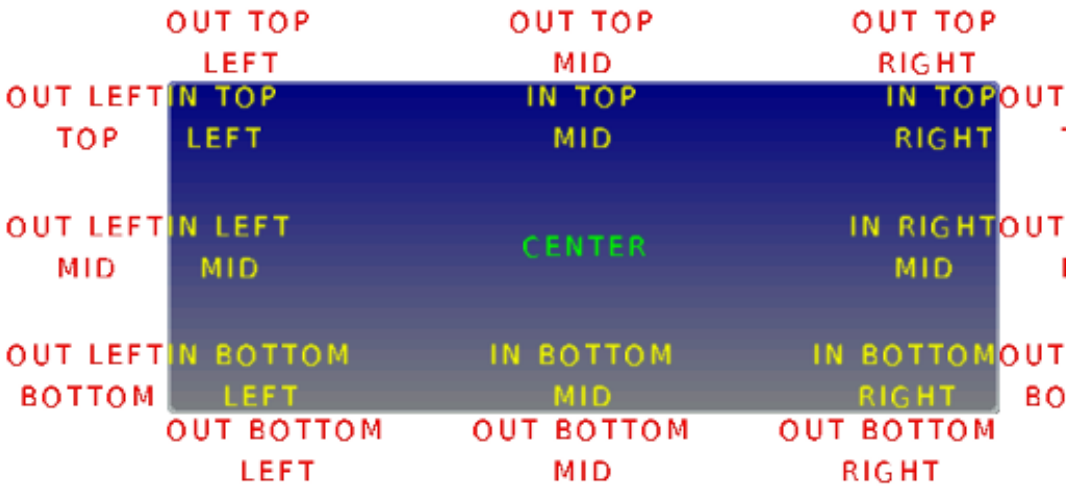
You can align the object on it's parent with `lv_obj_set_align(obj, LV_ALIGN_...)`. After this every x and y settings will be ralitiev to the set alignment mode. For example a this will shift the object by 10;20 px fro mthe center of its parent.

```
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 an object to an other 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(text, image, LV_ALIGN_OUT_BOTTOM_MID, 0, 10)`.



The following align types exists:

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 children of a parent use `lv_obj_get_child(parent, idx)`. Some examples for `idx`:

- `0` get the firstly (youngest) created child
- `1` get the secondly created child
- `-1` get the lastly (youngest) craeted child

The children can be iterated lke this

```
uint32_t i;
for(i = 0; i < lv_obj_get_child_cnt(parent); i++) {
  lv_obj_t * child = lv_obj_get_child(paernt, i);
  /*Do something with child*/
}
```

`lv_obj_get_child_id(obj)` tells the index of the object. That is how many younger children its parent has.

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)`.

Screens

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

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

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

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 the events.

Styles

Be sure to read the *Style overview*. Here or only the most essential functions are described.

A new style can be added to an object with `lv_obj_add_style(obj, &new_style, selector)` function. `selector` is a combination of part and state(s). E.g. `LV_PART_SCROLLBAR | LV_STATE_PRESSED`.

The Base object use `LV_PART_MAIN` style properties and `LV_PART_SCROLLBAR` with the typical backgroud style proeprties.

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 the 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 snapable 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_SNAPABLE` 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. consider rounded corners.
- `LV_OBJ_FLAG_IGNORE_LAYOUT` Make the object position-able 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 usersection.

Some examples:

```
/*Hide on object*/
lv_obj_add_flag(obj, LV_OBJ_FLAG_HIDDEN);

/*Make an obejct non-clickable*/
lv_obj_clear_flag(obj, LV_OBJ_FLAG_CLICKABLE);
```

Groups

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

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

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

Extended click area

By default, the objects can be clicked only on their coordinates, however, this area can be extended with `lv_obj_set_ext_click_area(obj, size)`.

5.1.2 Events

- `LV_EVENT_VALUE_CHANGED` when the `LV_OBJ_FLAG_CHECKABLE` flag is enabled and the obejct clicked (on transition to/from the checked state)

Learn more about *Events*.

5.1.3 Keys

If `LV_OBJ_FLAG_CHECKABLE` is enabled `LV_KEY_RIGHT` and `LV_KEY_UP` makes the object checked, and `LV_KEY_LEFT` and `LV_KEY_DOWN` makes it unchecked.

Learn more about *Keys*.

5.1.4 Example

5.1.5 API

Typedefs

typedef uint16_t **`lv_state_t`**

typedef uint32_t **`lv_part_t`**

typedef uint32_t **`lv_obj_flag_t`**

typedef struct *_lv_obj_t* **`lv_obj_t`**

Enums

enum **`[anonymous]`**

Possible states of a widget. OR-ed values are possible

Values:

enumerator **`LV_STATE_DEFAULT`**

enumerator **`LV_STATE_CHECKED`**

enumerator **`LV_STATE_FOCUSED`**

enumerator **`LV_STATE_FOCUS_KEY`**

enumerator **`LV_STATE_EDITED`**

enumerator **`LV_STATE_HOVERED`**

enumerator **`LV_STATE_PRESSED`**

enumerator **`LV_STATE_SCROLLED`**

enumerator **LV_STATE_DISABLED**

enumerator **LV_STATE_USER_1**

enumerator **LV_STATE_USER_2**

enumerator **LV_STATE_USER_3**

enumerator **LV_STATE_USER_4**

enumerator **LV_STATE_ANY**
Special value can be used in some functions to target all states

enum **[anonymous]**
The possible parts of widgets. The parts can be considered as the internal building block of the widgets. E.g. slider = background + indicator + knob Note every part is used by every widget

Values:

enumerator **LV_PART_MAIN**
A background like rectangle

enumerator **LV_PART_SCROLLBAR**
The scrollbar(s)

enumerator **LV_PART_INDICATOR**
Indicator, e.g. for slider, bar, switch, or the tick box of the checkbox

enumerator **LV_PART_KNOB**
Like handle to grab to adjust the value

enumerator **LV_PART_SELECTED**
Indicate the currently selected option or section

enumerator **LV_PART_ITEMS**
Used if the widget has multiple similar elements (e.g. tabel cells)

enumerator **LV_PART_TICKS**
Ticks on scale e.g. for a chart or meter

enumerator **LV_PART_CURSOR**
Mark a specific place e.g. for text area's cursor or on a chart

enumerator **LV_PART_CUSTOM_FIRST**
Extension point for custom widgets

enumerator **LV_PART_ANY**
Special value can be used in some functions to target all parts

enum **[anonymous]**
On/Off features controlling the object's behavior. OR-ed values are possible

Values:

enumerator **LV_OBJ_FLAG_HIDDEN**
Make the object hidden. (Like it wasn't there at all)

enumerator **LV_OBJ_FLAG_CLICKABLE**
Make the object clickable by the input devices

enumerator **LV_OBJ_FLAG_CLICK_FOCUSABLE**
Add focused state to the object when clicked

enumerator **LV_OBJ_FLAG_CHECKABLE**
Toggle checked state when the object is clicked

enumerator **LV_OBJ_FLAG_SCROLLABLE**
Make the object scrollable

enumerator **LV_OBJ_FLAG_SCROLL_ELASTIC**
Allow scrolling inside but with slower speed

enumerator **LV_OBJ_FLAG_SCROLL_MOMENTUM**
Make the object scroll further when "thrown"

enumerator **LV_OBJ_FLAG_SCROLL_ONE**
Allow scrolling only one snapable children

enumerator **LV_OBJ_FLAG_SCROLL_CHAIN**
Allow propagating the scroll to a parent

enumerator **LV_OBJ_FLAG_SCROLL_ON_FOCUS**
Automatically scroll object to make it visible when focused

enumerator **LV_OBJ_FLAG_SNAPABLE**
If scroll snap is enabled on the parent it can snap to this object

enumerator **LV_OBJ_FLAG_PRESS_LOCK**
Keep the object pressed even if the press slid from the object

enumerator **LV_OBJ_FLAG_EVENT_BUBBLE**
Propagate the events to the parent too

enumerator **LV_OBJ_FLAG_GESTURE_BUBBLE**
Propagate the gestures to the parent

enumerator **LV_OBJ_FLAG_ADV_HITTEST**
Allow performing more accurate hit (click) test. E.g. consider rounded corners.

enumerator **LV_OBJ_FLAG_IGNORE_LAYOUT**
Make the object position-able by the layouts

enumerator **LV_OBJ_FLAG_FLOATING**
Do not scroll the object when the parent scrolls and ignore layout

enumerator **LV_OBJ_FLAG_LAYOUT_1**

enumerator **LV_OBJ_FLAG_LAYOUT_2**
Custom flag, free to use by layouts

enumerator **LV_OBJ_FLAG_WIDGET_1**
Custom flag, free to use by layouts

enumerator **LV_OBJ_FLAG_WIDGET_2**
Custom flag, free to use by widget

enumerator **LV_OBJ_FLAG_USER_1**
Custom flag, free to use by widget

enumerator **LV_OBJ_FLAG_USER_2**
Custom flag, free to use by user

enumerator **LV_OBJ_FLAG_USER_3**
Custom flag, free to use by user

enumerator **LV_OBJ_FLAG_USER_4**
Custom flag, free to use by user

Functions

void **lv_init**(void)
Initialize LVGL library. Should be called before any other LVGL related function.

void **lv_deinit**(void)
Deinit the 'lv' library Currently only implemented when not using custom allocators, or GC is enabled.

lv_obj_t ***lv_obj_create**(*lv_obj_t* *parent)
Create a base object (a rectangle)

Parameters **parent** -- pointer to a parent object. If NULL then a screen will be created.

Returns pointer to the new object

void **lv_obj_add_flag**(*lv_obj_t* *obj, *lv_obj_flag_t* f)
Set one or more flags

Parameters

- **obj** -- pointer to an object
- **f** -- R-ed values from `lv_obj_flag_t` to set.

void **lv_obj_clear_flag**(*lv_obj_t* *obj, *lv_obj_flag_t* f)
Clear one or more flags

Parameters

- **obj** -- pointer to an object
- **f** -- OR-ed values from `lv_obj_flag_t` to set.

void **lv_obj_add_state**(*lv_obj_t* *obj, *lv_state_t* state)
Add one or more states to the object. The other state bits will remain unchanged. If specified in the styles, transition animation will be started from the previous state to the current.

Parameters

- **obj** -- pointer to an object
- **state** -- the states to add. E.g `LV_STATE_PRESSED | LV_STATE_FOCUSED`

void **lv_obj_clear_state**(*lv_obj_t* *obj, *lv_state_t* state)

Remove one or more states to the object. The other state bits will remain unchanged. If specified in the styles, transition animation will be started from the previous state to the current.

Parameters

- **obj** -- pointer to an object
- **state** -- the states to add. E.g `LV_STATE_PRESSED | LV_STATE_FOCUSED`

void **lv_obj_set_base_dir**(*lv_obj_t* *obj, lv_bidi_dir_t dir)

Set the base direction of the object

Parameters

- **obj** -- pointer to an object
- **dir** -- the new base direction. `LV_BIDI_DIR_LTR/RTL/AUTO/INHERIT`

static inline void **lv_obj_set_user_data**(*lv_obj_t* *obj, void *user_data)

Set the user_data field of the object

Parameters

- **obj** -- pointer to an object
- **user_data** -- pointer to the new user_data.

bool **lv_obj_has_flag**(const *lv_obj_t* *obj, *lv_obj_flag_t* f)

Check if a given flag or all the given flags are set on an object.

Parameters

- **obj** -- pointer to an object
- **f** -- the flag(s) to check (OR-ed values can be used)

Returns true: all flags are set; false: not all flags are set

bool **lv_obj_has_flag_any**(const *lv_obj_t* *obj, *lv_obj_flag_t* f)

Check if a given flag or any of the flags are set on an object.

Parameters

- **obj** -- pointer to an object
- **f** -- the flag(s) to check (OR-ed values can be used)

Returns true: at lest one flag flag is set; false: none of the flags are set

lv_bidi_dir_t **lv_obj_get_base_dir**(const *lv_obj_t* *obj)

Get the base direction of the object

Parameters **obj** -- pointer to an object

Returns the base direction. `LV_BIDI_DIR_LTR/RTL/AUTO/INHERIT`

lv_state_t **lv_obj_get_state**(const *lv_obj_t* *obj)

Get the state of an object

Parameters **obj** -- pointer to an object

Returns the state (OR-ed values from `lv_state_t`)

bool **lv_obj_has_state**(const *lv_obj_t* *obj, *lv_state_t* state)
Check if the object is in a given state or not.

Parameters

- **obj** -- pointer to an object
- **state** -- a state or combination of states to check

Returns true: `obj` is in `state`; false: `obj` is not in `state`

void ***lv_obj_get_group**(const *lv_obj_t* *obj)
Get the group of the object

Parameters **obj** -- pointer to an object

Returns the pointer to group of the object

static inline void ***lv_obj_get_user_data**(*lv_obj_t* *obj)
Get the user_data field of the object

Parameters **obj** -- pointer to an object

Returns the pointer to the user_data of the object

void **lv_obj_allocate_spec_attr**(*lv_obj_t* *obj)
Allocate special data for an object if not allocated yet.

Parameters **obj** -- pointer to an object

bool **lv_obj_check_type**(const *lv_obj_t* *obj, const lv_obj_class_t *class_p)
Get object's and its ancestors type. Put their name in `type_buf` starting with the current type. E.g. buf.type[0]="lv_btn", buf.type[1]="lv_cont", buf.type[2]="lv_obj"

Parameters

- **obj** -- pointer to an object which type should be get
- **buf** -- pointer to an `lv_obj_type_t` buffer to store the types

bool **lv_obj_has_class**(const *lv_obj_t* *obj, const lv_obj_class_t *class_p)
Check if any object has a given class (type). It checks the ancestor classes too.

Parameters

- **obj** -- pointer to an object
- **class_p** -- a class to check (e.g. `lv_slider_class`)

Returns true: `obj` has the given class

const lv_obj_class_t ***lv_obj_get_class**(const *lv_obj_t* *obj)
Get the class (type) of the object

Parameters **obj** -- pointer to an object

Returns the class (type) of the object

bool **lv_obj_is_valid**(const *lv_obj_t* *obj)
Check if any object is still "alive", and part of the hierarchy

Parameters

- **obj** -- pointer to an object
- **obj_type** -- type of the object. (e.g. "lv_btn")

Returns true: valid

static inline lv_coord_t **lv_obj_dpx**(const *lv_obj_t* *obj, lv_coord_t n)

Scale the given number of pixels (a distance or size) relative to a 160 DPI display considering the DPI of the `obj`'s display. It ensures that e.g. `lv_dpx(100)` will have the same physical size regardless to the DPI of the display.

Parameters

- **obj** -- an object whose display's dpi should be considered
- **n** -- the number of pixels to scale

Returns `n x current_dpi/160`

Variables

const lv_obj_class_t **lv_obj_class**

Make the base object's class publicly available.

struct **lv_obj_spec_attr_t**

#include <lv_obj.h> Special, rarely used attributes. They are allocated automatically if any elements is set.

Public Members

struct *_lv_obj_t* ****children**

Store the pointer of the children in an array.

uint32_t **child_cnt**

Number of children

lv_group_t ***group_p**

struct _lv_event_dsc_t ***event_dsc**

Dynamically allocated event callback and user data array

lv_point_t **scroll**

The current X/Y scroll offset

lv_coord_t **ext_click_pad**

Extra click padding in all direction

lv_coord_t **ext_draw_size**

EXTend the size in every direction for drawing.

lv_scrollbar_mode_t **scrollbar_mode**

How to display scrollbars

lv_scroll_snap_t **scroll_snap_x**

Where to align the snapable children horizontally

lv_scroll_snap_t **scroll_snap_y**

Where to align the snapable children horizontally

lv_dir_t **scroll_dir**
The allowed scroll direction(s)

lv_bidi_dir_t **base_dir**
Base direction of texts related to this object

uint8_t **event_dsc_cnt**
Number of event callabcks stored in `event_cb` array

struct **_lv_obj_t**

Public Members

const lv_obj_class_t ***class_p**

struct *_lv_obj_t* ***parent**

lv_obj_spec_attr_t ***spec_attr**

lv_obj_style_t ***styles**

void ***user_data**

lv_area_t **coords**

lv_obj_flag_t **flags**

lv_state_t **state**

uint16_t **layout_inv**

uint16_t **scr_layout_inv**

uint16_t **skip_trans**

uint16_t **style_cnt**

uint16_t **h_layout**

uint16_t **w_layout**

5.2 Core widgets

5.2.1 Arc (lv_arc)

Overview

The Arc are consists of a background and a foreground arc. The foregrond (indicator) arc can be adjusted by finger.

Parts and Styles

- `LV_PART_MAIN` It 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` It draws an other arc using the *arc* style properties. It's padding values are interpreted relative to the background arc.
- `LV_PART_KNOB`It draws a handle on the end of the indicator. It uses all background properties and padding values. With zero padding the knob size is the same as the indicator's width. Larger padding makes it larger, smaller padding makes it smaller.

Usage

Value and range

A new value can be set by `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 1..100.

The indicator arc is drawn on the main part's arc. That is if the vale is set to maximum the indicator arc will cover the entire "background" arc. To set the start and end angl of the background arc use the `lv_arc_set_bg_angles(arc, start_angle, end_angle)` function or `lv_arc_set_bg_start/end_angle(arc, start_angle)`.

Zero degree is at the middle right (3 o'clock) of the object and the degrees are increasing in clockwise direction. The angles should be in [0;360] range.

Rotation

An offset to the 0 degree position can 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 minimimum 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 the angle is set by `lv_arc_set_value()` or the arc is adjusted by finger.

Change rate

If the 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_rage(arc, rate)`

Setting the indicator manually

It also possible to set the angles o 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)` sets the angles of the indicator arc. In this case the set "value" and "mode" is ignored.

In other words, settings angles and values are independent. You should use either value and angle settings. Mixing the two might result unintended behavior.

To make the arc non-adjutabe 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);
```

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 for the background rectangle, the background arc, the foreground arc and the knob to allow hooking the drawing. For more detail on the background rectangle part see the Base object's documentation. The fields of `lv_obj_draw_dsc_t` is set like the followings:
  - For both arcs: `clip_area`, `p1` (center of the arc), `radius`, `arc_dsc`, `part`.
  - For the knob: `clip_area`, `draw_area`, `rect_dsc`, `part`.

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

C

Simple Arc

code

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

#if LV_USE_ARC && LV_BUILD_EXAMPLES

void lv_example_arc_1(void)
{
  /*Create an Arc*/
  lv_obj_t * arc = lv_arc_create(lv_scr_act());
  lv_obj_set_size(arc, 150, 150);
  lv_arc_set_rotation(arc, 135);
```

(continues on next page)

(continued from previous page)

```
  lv_arc_set_bg_angles(arc, 0, 270);
  lv_arc_set_value(arc, 40);
  lv_obj_center(arc);
}

#endif
```

Loader with Arc

code

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

#if LV_USE_ARC && LV_BUILD_EXAMPLES

static void set_angle(void * obj, int32_t v)
{
    lv_arc_set_value(obj, v);
}

/**
 * Create an arc which acts as a loader.
 */
void lv_example_arc_2(void)
{
  /*Create an Arc*/
  lv_obj_t * arc = lv_arc_create(lv_scr_act());
  lv_arc_set_rotation(arc, 270);
  lv_arc_set_bg_angles(arc, 0, 360);
  lv_obj_remove_style(arc, NULL, LV_PART_KNOB);   /*Be sure the knob is not␣
↪displayed*/
  lv_obj_clear_flag(arc, LV_OBJ_FLAG_CLICKABLE);  /*To not allow adjusting by click*/
  lv_obj_center(arc);

  lv_anim_t a;
  lv_anim_init(&a);
  lv_anim_set_var(&a, arc);
  lv_anim_set_exec_cb(&a, set_angle);
  lv_anim_set_time(&a, 1000);
  lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);    /*Just for the demo*/
  lv_anim_set_repeat_delay(&a, 500);
  lv_anim_set_values(&a, 0, 100);
  lv_anim_start(&a);



}

#endif
```

MicroPython

No examples yet.

API

Typedefs

typedef uint8_t **lv_arc_mode_t**

Enums

enum **[anonymous]**
Values:

enumerator **LV_ARC_MODE_NORMAL**

enumerator **LV_ARC_MODE_SYMMETRICAL**

enumerator **LV_ARC_MODE_REVERSE**

Functions

lv_obj_t ***lv_arc_create**(*lv_obj_t* *parent)
Create a arc objects

Parameters **par** -- pointer to an object, it will be the parent of the new arc

Returns pointer to the created arc

void **lv_arc_set_start_angle**(*lv_obj_t* *arc, uint16_t start)
Set the start angle of an arc. 0 deg: right, 90 bottom, etc.

Parameters

- **arc** -- pointer to an arc object
- **start** -- the start angle

void **lv_arc_set_end_angle**(*lv_obj_t* *arc, uint16_t end)
Set the end angle of an arc. 0 deg: right, 90 bottom, etc.

Parameters

- **arc** -- pointer to an arc object
- **end** -- the end angle

void **lv_arc_set_angles**(*lv_obj_t* *arc, uint16_t start, uint16_t end)
Set the start and end angles

Parameters

- **arc** -- pointer to an arc object
- **start** -- the start angle
- **end** -- the end angle

void **lv_arc_set_bg_start_angle**(*lv_obj_t* *arc, uint16_t start)
Set the start angle of an arc background. 0 deg: right, 90 bottom, etc.

Parameters

- **arc** -- pointer to an arc object
- **start** -- the start angle

void **lv_arc_set_bg_end_angle**(*lv_obj_t* *arc, uint16_t end)
Set the start angle of an arc background. 0 deg: right, 90 bottom etc.

Parameters

- **arc** -- pointer to an arc object
- **end** -- the end angle

void **lv_arc_set_bg_angles**(*lv_obj_t* *arc, uint16_t start, uint16_t end)
Set the start and end angles of the arc background

Parameters

- **arc** -- pointer to an arc object
- **start** -- the start angle
- **end** -- the end angle

void **lv_arc_set_rotation**(*lv_obj_t* *arc, uint16_t rotation)
Set the rotation for the whole arc

Parameters

- **arc** -- pointer to an arc object
- **rotation** -- rotation angle

void **lv_arc_set_mode**(*lv_obj_t* *arc, *lv_arc_mode_t* type)
Set the type of arc.

Parameters

- **arc** -- pointer to arc object
- **mode** -- arc's mode

void **lv_arc_set_value**(*lv_obj_t* *arc, int16_t value)
Set a new value on the arc

Parameters

- **arc** -- pointer to a arc object
- **value** -- new value

void **lv_arc_set_range**(*lv_obj_t* *arc, int16_t min, int16_t max)
Set minimum and the maximum values of a arc

Parameters

- **arc** -- pointer to the arc object
- **min** -- minimum value
- **max** -- maximum value

void **lv_arc_set_change_rate**(*lv_obj_t* *arc, uint16_t rate)
Set a change rate to limit the speed how fast the arc should reache the pressed point.

Parameters

- **arc** -- pointer to a arc object
- **rate** -- the change rate

uint16_t **lv_arc_get_angle_start**(*lv_obj_t* *obj)
Get the start angle of an arc.

Parameters **arc** -- pointer to an arc object

Returns the start angle [0..360]

uint16_t **lv_arc_get_angle_end**(*lv_obj_t* *obj)
Get the end angle of an arc.

Parameters **arc** -- pointer to an arc object

Returns the end angle [0..360]

uint16_t **lv_arc_get_bg_angle_start**(*lv_obj_t* *obj)
Get the start angle of an arc background.

Parameters **arc** -- pointer to an arc object

Returns the start angle [0..360]

uint16_t **lv_arc_get_bg_angle_end**(*lv_obj_t* *obj)
Get the end angle of an arc background.

Parameters **arc** -- pointer to an arc object

Returns the end angle [0..360]

int16_t **lv_arc_get_value**(const *lv_obj_t* *obj)
Get the value of a arc

Parameters **arc** -- pointer to a arc object

Returns the value of the arc

int16_t **lv_arc_get_min_value**(const *lv_obj_t* *obj)
Get the minimum value of a arc

Parameters **arc** -- pointer to a arc object

Returns the minimum value of the arc

int16_t **lv_arc_get_max_value**(const *lv_obj_t* *obj)
Get the maximum value of a arc

Parameters **arc** -- pointer to a arc object

Returns the maximum value of the arc

lv_arc_mode_t **lv_arc_get_mode**(const *lv_obj_t* *obj)
Get whether the arc is type or not.

Parameters **arc** -- pointer to a arc object

Returns arc's mode

Variables

const lv_obj_class_t **lv_arc_class**

struct **lv_arc_t**

Public Members

lv_obj_t **obj**

uint16_t **rotation**

uint16_t **indic_angle_start**

uint16_t **indic_angle_end**

uint16_t **bg_angle_start**

uint16_t **bg_angle_end**

int16_t **value**

int16_t **min_value**

int16_t **max_value**

uint16_t **dragging**

uint16_t **type**

uint16_t **min_close**

uint16_t **chg_rate**

uint32_t **last_tick**

int16_t **last_angle**

5.2.2 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 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 and it also 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 1..100.

The new value in `lv_bar_set_value` can be set with or without an animation depending on the last parameter (`LV_ANIM_ON/OFF`).

Modes

The bar can be one the following modes:

- `LV_BAR_MODE_NORMAL` A normal bar as described above
- `LV_BAR_SYMMETRICAL` Draw the indicator form the zero value to current value. Requires negaitve 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 has to be always smaller than the end value.

Events

- `LV_EVENT_DRAW_PART_BEGIN` and `LV_EVENT_DRAW_PART_END` are sent for both main and indicator parts to allow hooking the drawing. The for more detail on the main part see the Base object's documentation. For the indicator the following fields are used: `clip_area`, `draw_area`, `rect_dsc`, `part`.

Learn more about *Events*.

Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

Example

C

Simple Bar

code

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

void lv_example_bar_1(void)
{
    lv_obj_t * bar1 = lv_bar_create(lv_scr_act());
    lv_obj_set_size(bar1, 200, 20);
    lv_obj_center(bar1);
    lv_bar_set_value(bar1, 70, LV_ANIM_OFF);
}

#endif
```

Styling a bar

code

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

/**
 * Example of styling the bar
 */
void lv_example_bar_2(void)
{
    static lv_style_t style_bg;
    static lv_style_t style_indic;

    lv_style_init(&style_bg);
    lv_style_set_border_color(&style_bg, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_border_width(&style_bg, 2);
    lv_style_set_pad_all(&style_bg, 6); /*To make the indicator smaller*/
    lv_style_set_radius(&style_bg, 6);
    lv_style_set_anim_time(&style_bg, 1000);

    lv_style_init(&style_indic);
    lv_style_set_bg_opa(&style_indic, LV_OPA_COVER);
    lv_style_set_bg_color(&style_indic, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_radius(&style_indic, 3);

    lv_obj_t * bar = lv_bar_create(lv_scr_act());
    lv_obj_remove_style_all(bar);  /*To have a clean start*/
    lv_obj_add_style(bar, &style_bg, 0);
    lv_obj_add_style(bar, &style_indic, LV_PART_INDICATOR);

    lv_obj_set_size(bar, 200, 20);
    lv_obj_center(bar);
```

(continues on next page)

(continued from previous page)

```
    lv_bar_set_value(bar, 100, LV_ANIM_ON);
}

#endif
```

Temperature meter

code

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

static void set_temp(void * bar, int32_t temp)
{
    lv_bar_set_value(bar, temp, LV_ANIM_ON);
}

/**
 * A temperature meter example
 */
void lv_example_bar_3(void)
{
    static lv_style_t style_indic;

    lv_style_init(&style_indic);
    lv_style_set_bg_opa(&style_indic, LV_OPA_COVER);
    lv_style_set_bg_color(&style_indic, lv_palette_main(LV_PALETTE_RED));
    lv_style_set_bg_grad_color(&style_indic, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_bg_grad_dir(&style_indic, LV_GRAD_DIR_VER);

    lv_obj_t * bar = lv_bar_create(lv_scr_act());
    lv_obj_add_style(bar, &style_indic, LV_PART_INDICATOR);
    lv_obj_set_size(bar, 20, 200);
    lv_obj_center(bar);
    lv_bar_set_range(bar, -20, 40);

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_exec_cb(&a, set_temp);
    lv_anim_set_time(&a, 3000);
    lv_anim_set_playback_time(&a, 3000);
    lv_anim_set_var(&a, bar);
    lv_anim_set_values(&a, -20, 40);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);
    lv_anim_start(&a);
}


#endif
```

Stripe pattern and range value

code

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

/**
 * Bar with stripe pattern and ranged value
 */
void lv_example_bar_4(void)
{
    LV_IMG_DECLARE(img_skew_strip);
    static lv_style_t style_indic;

    lv_style_init(&style_indic);
    lv_style_set_bg_img_src(&style_indic, &img_skew_strip);
    lv_style_set_bg_img_tiled(&style_indic, true);
    lv_style_set_bg_img_opa(&style_indic, LV_OPA_30);

    lv_obj_t * bar = lv_bar_create(lv_scr_act());
    lv_obj_add_style(bar, &style_indic, LV_PART_INDICATOR);

    lv_obj_set_size(bar, 260, 20);
    lv_obj_center(bar);
    lv_bar_set_mode(bar, LV_BAR_MODE_RANGE);
    lv_bar_set_value(bar, 90, LV_ANIM_OFF);
    lv_bar_set_start_value(bar, 20, LV_ANIM_OFF);
}

#endif
```

Bar with RTL and RTL base direction

code

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

/**
 * Bar with LTR and RTL base direction
 */
void lv_example_bar_5(void)
{
    lv_obj_t * label;


    lv_obj_t * bar_ltr = lv_bar_create(lv_scr_act());
    lv_obj_set_size(bar_ltr, 200, 20);
    lv_bar_set_value(bar_ltr, 70, LV_ANIM_OFF);
    lv_obj_align(bar_ltr, LV_ALIGN_CENTER, 0, -30);

    label = lv_label_create(lv_scr_act());
    lv_label_set_text(label, "Left to Right base direction");
    lv_obj_align_to(label, bar_ltr, LV_ALIGN_OUT_TOP_MID, 0, -5);
```

(continues on next page)

(continued from previous page)

```
    lv_obj_t * bar_rtl = lv_bar_create(lv_scr_act());
    lv_obj_set_base_dir(bar_rtl, LV_BIDI_DIR_RTL);
    lv_obj_set_size(bar_rtl, 200, 20);
    lv_bar_set_value(bar_rtl, 70, LV_ANIM_OFF);
    lv_obj_align(bar_rtl, LV_ALIGN_CENTER, 0, 30);

    label = lv_label_create(lv_scr_act());
    lv_label_set_text(label, "Right to Left base direction");
    lv_obj_align_to(label, bar_rtl, LV_ALIGN_OUT_TOP_MID, 0, -5);
}

#endif
```

Custom drawr to show the current value

code

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

static void set_value(void *bar, int32_t v)
{
    lv_bar_set_value(bar, v, LV_ANIM_OFF);
}

static void event_cb(lv_event_t * e)
{
    lv_obj_draw_part_dsc_t * dsc = lv_event_get_param(e);
    if(dsc->part != LV_PART_INDICATOR) return;

    lv_obj_t * obj= lv_event_get_target(e);

    lv_draw_label_dsc_t label_dsc;
    lv_draw_label_dsc_init(&label_dsc);
    label_dsc.font = LV_FONT_DEFAULT;

    char buf[8];
    lv_snprintf(buf, sizeof(buf), "%d", lv_bar_get_value(obj));

    lv_point_t txt_size;
    lv_txt_get_size(&txt_size, buf, label_dsc.font, label_dsc.letter_space, label_dsc.
↪line_space, LV_COORD_MAX, label_dsc.flag);

    lv_area_t txt_area;
    /*If the indicator is long enough put the text inside on the right*/
    if(lv_area_get_width(dsc->draw_area) > txt_size.x + 20) {
        txt_area.x2 = dsc->draw_area->x2 - 5;
        txt_area.x1 = txt_area.x2 - txt_size.x + 1;
        label_dsc.color = lv_color_white();
    }
    /*If the indicator is still short put the text out of it on the right*/
    else {
        txt_area.x1 = dsc->draw_area->x2 + 5;
        txt_area.x2 = txt_area.x1 + txt_size.x - 1;
        label_dsc.color = lv_color_black();
```

(continues on next page)

(continued from previous page)

```
    }

    txt_area.y1 = dsc->draw_area->y1 + (lv_area_get_height(dsc->draw_area) - txt_size.
↪y) / 2;
    txt_area.y2 = txt_area.y1 + txt_size.y - 1;

    lv_draw_label(&txt_area, dsc->clip_area, &label_dsc, buf, NULL);
}

/**
 * Custom drawer on the bar to display the current value
 */
void lv_example_bar_6(void)
{
    lv_obj_t * bar = lv_bar_create(lv_scr_act());
    lv_obj_add_event_cb(bar, event_cb, LV_EVENT_DRAW_PART_END, NULL);
    lv_obj_set_size(bar, 200, 20);
    lv_obj_center(bar);

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, bar);
    lv_anim_set_values(&a, 0, 100);
    lv_anim_set_exec_cb(&a, set_value);
    lv_anim_set_time(&a, 2000);
    lv_anim_set_playback_time(&a, 2000);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);
    lv_anim_start(&a);

}

#endif
```

MicroPython

No examples yet.

API

Typedefs

typedef uint8_t **`lv_bar_mode_t`**

Enums

enum **[anonymous]**
Values:

enumerator **LV_BAR_MODE_NORMAL**

enumerator **LV_BAR_MODE_SYMMETRICAL**

enumerator **LV_BAR_MODE_RANGE**

Functions

lv_obj_t ***lv_bar_create**(*lv_obj_t* *parent)
Create a bar objects

Parameters **parent** -- pointer to an object, it will be the parent of the new bar

Returns pointer to the created bar

void **lv_bar_set_value**(*lv_obj_t* *obj, int32_t value, *lv_anim_enable_t* anim)
Set a new value on the bar

Parameters

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

void **lv_bar_set_start_value**(*lv_obj_t* *obj, int32_t start_value, *lv_anim_enable_t* anim)
Set a new start value on the bar

Parameters

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

void **lv_bar_set_range**(*lv_obj_t* *obj, int32_t min, int32_t max)
Set minimum and the maximum values of a bar

Parameters

- **obj** -- pointer to the bar object
- **min** -- minimum value
- **max** -- maximum value

void **lv_bar_set_mode**(*lv_obj_t* *obj, *lv_bar_mode_t* mode)
Set the type of bar.

Parameters

- **obj** -- pointer to bar object
- **mode** -- bar type from ::lv_bar_mode_t

int32_t **lv_bar_get_value**(const *lv_obj_t* *obj)
Get the value of a bar

Parameters **obj** -- pointer to a bar object

Returns the value of the bar

int32_t **lv_bar_get_start_value**(const *lv_obj_t* *obj)
Get the start value of a bar

Parameters **obj** -- pointer to a bar object

Returns the start value of the bar

int32_t **lv_bar_get_min_value**(const *lv_obj_t* *obj)
Get the minimum value of a bar

Parameters **obj** -- pointer to a bar object

Returns the minimum value of the bar

int32_t **lv_bar_get_max_value**(const *lv_obj_t* *obj)
Get the maximum value of a bar

Parameters **obj** -- pointer to a bar object

Returns the maximum value of the bar

lv_bar_mode_t **lv_bar_get_mode**(*lv_obj_t* *obj)
Get the type of bar.

Parameters **obj** -- pointer to bar object

Returns bar type from ::lv_bar_mode_t

Variables

const lv_obj_class_t **lv_bar_class**

struct **lv_bar_anim_t**

Public Members

lv_obj_t ***bar**

int32_t **anim_start**

int32_t **anim_end**

int32_t **anim_state**

struct **lv_bar_t**

Public Members

lv_obj_t **obj**

int32_t **cur_value**
Current value of the bar

int32_t **min_value**
Minimum value of the bar

int32_t **max_value**
Maximum value of the bar

int32_t **start_value**
Start value of the bar

lv_area_t **indic_area**
Save the indicator area. Might be used by derived types

lv_bar_anim_t **cur_value_anim**

lv_bar_anim_t **start_value_anim**

lv_bar_mode_t **mode**
Type of bar

5.2.3 Button (lv_btn)

Overview

Buttons has no new features compared to the *Base object*. It usufule for semantic purposes and has slightly different default settings.

Buttons differ from Base object in the following points by default:

- Not scrollable
- Added to the default group
- Its default height and width is `LV_SIZE_CONTENT`

Parts and Styles

- `LV_PART_MAIN` The background of the button. It 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 obejct clicked (on transition to/from the checked state)

Learn more about *Events*.

Keys

If `LV_OBJ_FLAG_CHECKABLE` is enabled `LV_KEY_RIGHT` and `LV_KEY_UP` makes the object checked, and `LV_KEY_LEFT` and `LV_KEY_DOWN` makes it unchecked.

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

Learn more about *Keys*.

Example

C

Simple Buttons

code

```
#include "../../lv_examples.h"
#if LV_USE_BTN && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);

    if(code == LV_EVENT_CLICKED) {
        LV_LOG_USER("Clicked");
    }
    else if(code == LV_EVENT_VALUE_CHANGED) {
        LV_LOG_USER("Toggled");
    }
}

void lv_example_btn_1(void)
{
    lv_obj_t * label;

    lv_obj_t * btn1 = lv_btn_create(lv_scr_act());
    lv_obj_add_event_cb(btn1, event_handler, LV_EVENT_ALL, NULL);
    lv_obj_align(btn1, LV_ALIGN_CENTER, 0, -40);

    label = lv_label_create(btn1);
    lv_label_set_text(label, "Button");
    lv_obj_center(label);
```

(continues on next page)

(continued from previous page)

```
    lv_obj_t * btn2 = lv_btn_create(lv_scr_act());
    lv_obj_add_event_cb(btn2, event_handler, LV_EVENT_ALL, NULL);
    lv_obj_align(btn2, LV_ALIGN_CENTER, 0, 40);
    lv_obj_add_flag(btn2, LV_OBJ_FLAG_CHECKABLE);
    lv_obj_set_height(btn2, LV_SIZE_CONTENT);

    label = lv_label_create(btn2);
    lv_label_set_text(label, "Toggle");
    lv_obj_center(label);
}
#endif
```

Styling buttons

code

```
#include "../../lv_examples.h"
#if LV_USE_BTN && LV_BUILD_EXAMPLES

/**
 * Style a button from scratch
 */
void lv_example_btn_2(void)
{
    /*Init the style for the default state*/
    static lv_style_t style;
    lv_style_init(&style);

    lv_style_set_radius(&style, 3);

    lv_style_set_bg_opa(&style, LV_OPA_100);
    lv_style_set_bg_color(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_bg_grad_color(&style, lv_palette_darken(LV_PALETTE_BLUE, 2));
    lv_style_set_bg_grad_dir(&style, LV_GRAD_DIR_VER);

    lv_style_set_border_opa(&style, LV_OPA_40);
    lv_style_set_border_width(&style, 2);
    lv_style_set_border_color(&style, lv_palette_main(LV_PALETTE_GREY));

    lv_style_set_shadow_width(&style, 8);
    lv_style_set_shadow_color(&style, lv_palette_main(LV_PALETTE_GREY));
    lv_style_set_shadow_ofs_y(&style, 8);

    lv_style_set_outline_opa(&style, LV_OPA_COVER);
    lv_style_set_outline_color(&style, lv_palette_main(LV_PALETTE_BLUE));

    lv_style_set_text_color(&style, lv_color_white());
    lv_style_set_pad_all(&style, 10);

    /*Init the pressed style*/
    static lv_style_t style_pr;
    lv_style_init(&style_pr);

    /*Ad a large outline when pressed*/
```

(continues on next page)

(continued from previous page)

```
    lv_style_set_outline_width(&style_pr, 30);
    lv_style_set_outline_opa(&style_pr, LV_OPA_TRANSP);

    lv_style_set_translate_y(&style_pr, 5);
    lv_style_set_shadow_ofs_y(&style_pr, 3);
    lv_style_set_bg_color(&style_pr, lv_palette_darken(LV_PALETTE_BLUE, 2));
    lv_style_set_bg_grad_color(&style_pr, lv_palette_darken(LV_PALETTE_BLUE, 4));

    /*Add a transition to the the outline*/
    static lv_style_transition_dsc_t trans;
    static lv_style_prop_t props[] = {LV_STYLE_OUTLINE_WIDTH, LV_STYLE_OUTLINE_OPA, 0}
↪;
    lv_style_transition_dsc_init(&trans, props, lv_anim_path_linear, 300, 0, NULL);

    lv_style_set_transition(&style_pr, &trans);

    lv_obj_t * btn1 = lv_btn_create(lv_scr_act());
    lv_obj_remove_style_all(btn1);                          /*Remove the style coming␣
↪from the theme*/
    lv_obj_add_style(btn1, &style, 0);
    lv_obj_add_style(btn1, &style_pr, LV_STATE_PRESSED);
    lv_obj_set_size(btn1, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_center(btn1);

    lv_obj_t * label = lv_label_create(btn1);
    lv_label_set_text(label, "Button");
    lv_obj_center(label);
}
#endif
```

Gummy button

code

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_BTN

/**
 * Create a style transition on a button to act like a gum when clicked
 */
void lv_example_btn_3(void)
{
    /*Properties to transition*/
    static lv_style_prop_t props[] = {
            LV_STYLE_TRANSFORM_WIDTH, LV_STYLE_TRANSFORM_HEIGHT, LV_STYLE_TEXT_LETTER_
↪SPACE, 0
    };

    /*Transition descriptor when going back to the default state.
     *Add some delay to be sure the press transition is visible even if the press was␣
↪very short*/
    static lv_style_transition_dsc_t transition_dsc_def;
    lv_style_transition_dsc_init(&transition_dsc_def, props, lv_anim_path_overshoot,␣
↪250, 100, NULL);
```

(continues on next page)

(continued from previous page)

```
    /*Transition descriptor when going to pressed state.
     *No delay, go to presses state immediately*/
    static lv_style_transition_dsc_t transition_dsc_pr;
    lv_style_transition_dsc_init(&transition_dsc_pr, props, lv_anim_path_ease_in_out,␣
↪250, 0, NULL);

    /*Add only the new transition to he default state*/
    static lv_style_t style_def;
    lv_style_init(&style_def);
    lv_style_set_transition(&style_def, &transition_dsc_def);

    /*Add the transition and some transformation to the presses state.*/
    static lv_style_t style_pr;
    lv_style_init(&style_pr);
    lv_style_set_transform_width(&style_pr, 10);
    lv_style_set_transform_height(&style_pr, -10);
    lv_style_set_text_letter_space(&style_pr, 10);
    lv_style_set_transition(&style_pr, &transition_dsc_pr);

    lv_obj_t * btn1 = lv_btn_create(lv_scr_act());
    lv_obj_align(btn1, LV_ALIGN_CENTER, 0, -80);
    lv_obj_add_style(btn1, &style_pr, LV_STATE_PRESSED);
    lv_obj_add_style(btn1, &style_def, 0);

    lv_obj_t * label = lv_label_create(btn1);
    lv_label_set_text(label, "Gum");
}
#endif
```

MicroPython

No examples yet.

API

Functions

lv_obj_t ***lv_btn_create**(*lv_obj_t* *parent)

Create a button object

Parameters **parent** -- pointer to an object, it will be the parent of the new button

Returns pointer to the created button

Variables

const lv_obj_class_t **lv_btn_class**

struct **lv_btn_t**

Public Members

lv_obj_t **obj**

5.2.4 Button matrix (lv_btnmatrix)

Overview

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

The Button matrix object is very light weighted because the buttons are not created just virtually drawn on the fly. This way, 1 button use only 8 extra bytes instead of the ~100-150 byte size of a normal *Button* object and other ~100 byte for the size of the *Label* object.

The Button matrix is added to the deafult group (if it is set). Besides the Button matrix is an editable object to allow selecting and clicing the buttons with encoder navigation too.

Parts and Styles

- `LV_PART_MAIN` The bacground of the button matrix. It uses the typical background style properties. `pad_row` and `pad_column` sets the space between the buttons.
- `LV_PART_ITEMS` The buttons and they all use the text and typical background style properties expect 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 `NULL` or an empty string (`""`)!

Use `"\n"` in the map to make **line break**. E.g. `{"btn1", "btn2", "\n", "btn3", ""}`. Each line's buttons have their width calculated automatically. 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's value mus be in the [1..7] range and the deafult 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_CHECHED` will be added/removed as the button is clicked
- `LV_BTNMATRIX_CTRL_CHECKED` MAke the button checked. It will use the `LV_STATE_CHECHKED` 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_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 deafult 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, btn_id, LV_BTNM_CTRL_...)` and `lv_btnmatrix_clear_btn_ctrl_all(btnm, btn_id, LV_BTNMATRIX_CTRL_...)`.

The set a control map for a button matrix (similarly to the map for the text), use `lv_btnmatrix_set_ctrl_map(btnm, ctrl_map)`. An element of `ctrl_map` should look like `ctrl_map[0] = width | LV_BTNM_CTRL_NO_REPEAT | LV_BTNM_CTRL_CHECHKABLE`. 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 once.

Events

- `LV_EVENT_VALUE_CHANGED` Sent when a button is pressed/released or repeated after long press. The event paramter is set to the ID of the pressed/released button.
- `LV_EVENT_DRAW_PART_BEGIN` and `LV_EVENT_DRAW_PART_END` are sent for both the main and the items (buttons) parts to allow hooking the drawing. The for more detail on the main part see the Base object's documentation. For the buttons the following fields are used: `clip_area`, `draw_area`, `rect_dsc`, `rect_dsc`, `part`, `id` (index of the button being drawn).

`lv_btnmatrix_get_selected_btn(btnm)` returns the index of the lastly pressed, 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

C

Simple Button matrix

code

```
#include "../../lv_examples.h"
#if LV_USE_BTNMATRIX && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        uint32_t id = lv_btnmatrix_get_selected_btn(obj);
        const char * txt = lv_btnmatrix_get_btn_text(obj, id);

        LV_LOG_USER("%s was pressed\n", txt);
    }
}


static const char * btnm_map[] = {"1", "2", "3", "4", "5", "\n",
                                  "6", "7", "8", "9", "0", "\n",
                                  "Action1", "Action2", ""};

void lv_example_btnmatrix_1(void)
{
    lv_obj_t * btnm1 = lv_btnmatrix_create(lv_scr_act());
```

(continues on next page)

(continued from previous page)

```
    lv_btnmatrix_set_map(btnm1, btnm_map);
    lv_btnmatrix_set_btn_width(btnm1, 10, 2);        /*Make "Action1" twice as wide␣
↪as "Action2"*/
    lv_btnmatrix_set_btn_ctrl(btnm1, 10, LV_BTNMATRIX_CTRL_CHECKABLE);
    lv_btnmatrix_set_btn_ctrl(btnm1, 11, LV_BTNMATRIX_CTRL_CHECKED);
    lv_obj_align(btnm1, LV_ALIGN_CENTER, 0, 0);
    lv_obj_add_event_cb(btnm1, event_handler, LV_EVENT_ALL, NULL);
}

#endif
```

Custom buttons

code

```
#include "../../lv_examples.h"
#if LV_USE_BTNMATRIX && LV_BUILD_EXAMPLES


static void event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target(e);
    if(code == LV_EVENT_DRAW_PART_BEGIN) {
        lv_obj_draw_part_dsc_t * dsc = lv_event_get_param(e);

        /*Change the draw descriptor the 2nd button*/
        if(dsc->id == 1) {
            dsc->rect_dsc->radius = 0;
            if(lv_btnmatrix_get_selected_btn(obj) == dsc->id)  dsc->rect_dsc->bg_
↪color = lv_palette_darken(LV_PALETTE_GREY, 3);
            else dsc->rect_dsc->bg_color = lv_palette_main(LV_PALETTE_BLUE);

            dsc->rect_dsc->shadow_width = 6;
            dsc->rect_dsc->shadow_ofs_x = 3;
            dsc->rect_dsc->shadow_ofs_y = 3;
            dsc->label_dsc->color = lv_color_white();
        }
        /*Change the draw descriptor the 3rd button*/
        else if(dsc->id == 2) {
            dsc->rect_dsc->radius = LV_RADIUS_CIRCLE;
            if(lv_btnmatrix_get_selected_btn(obj) == dsc->id)  dsc->rect_dsc->bg_
↪color = lv_palette_darken(LV_PALETTE_RED, 3);
            else dsc->rect_dsc->bg_color = lv_palette_main(LV_PALETTE_RED);

            dsc->label_dsc->color = lv_color_white();
        }
        else if(dsc->id == 3) {
            dsc->label_dsc->opa = LV_OPA_TRANSP; /*Hide the text if any*/

        }
    }
    if(code == LV_EVENT_DRAW_PART_END) {
        lv_obj_draw_part_dsc_t * dsc = lv_event_get_param(e);
```

(continues on next page)

(continued from previous page)

```
        /*Add custom content to the 4th button when the button itself was drawn*/
        if(dsc->id == 3) {
            LV_IMG_DECLARE(img_star);
            lv_img_header_t header;
            lv_res_t res = lv_img_decoder_get_info(&img_star, &header);
            if(res != LV_RES_OK) return;

            lv_area_t a;
            a.x1 = dsc->draw_area->x1 + (lv_area_get_width(dsc->draw_area) - header.
↪w) / 2;
            a.x2 = a.x1 + header.w - 1;
            a.y1 = dsc->draw_area->y1 + (lv_area_get_height(dsc->draw_area) - header.
↪h) / 2;
            a.y2 = a.y1 + header.h - 1;

            lv_draw_img_dsc_t img_draw_dsc;
            lv_draw_img_dsc_init(&img_draw_dsc);
            img_draw_dsc.recolor = lv_color_black();
            if(lv_btnmatrix_get_selected_btn(obj) == dsc->id)  img_draw_dsc.recolor_
↪opa = LV_OPA_30;

            lv_draw_img(&a, dsc->clip_area, &img_star, &img_draw_dsc);
        }
    }
}

/**
 * Add custom drawer to the button matrix to customize butons one by one
 */
void lv_example_btnmatrix_2(void)
{
    lv_obj_t * btnm = lv_btnmatrix_create(lv_scr_act());
    lv_obj_add_event_cb(btnm, event_cb, LV_EVENT_ALL, NULL);
    lv_obj_center(btnm);
}

#endif
```

Pagination

code

```
#include "../../lv_examples.h"
#if LV_USE_BTNMATRIX  && LV_BUILD_EXAMPLES

static void event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target(e);
    uint32_t id = lv_btnmatrix_get_selected_btn(obj);
    bool prev = id == 0 ? true : false;
    bool next = id == 6 ? true : false;
    if(prev || next) {
        /*Find the checked button*/
        uint32_t i;
        for(i = 1; i < 7; i++) {
```

(continues on next page)

(continued from previous page)

```
            if(lv_btnmatrix_has_btn_ctrl(obj, i, LV_BTNMATRIX_CTRL_CHECKED)) break;
        }

        if(prev && i > 1) i--;
        else if(next && i < 5) i++;

        lv_btnmatrix_set_btn_ctrl(obj, i, LV_BTNMATRIX_CTRL_CHECKED);
    }
}

/**
 * Make a button group (pagination)
 */
void lv_example_btnmatrix_3(void)
{
    static lv_style_t style_bg;
    lv_style_init(&style_bg);
    lv_style_set_pad_all(&style_bg, 0);
    lv_style_set_pad_gap(&style_bg, 0);
    lv_style_set_clip_corner(&style_bg, true);
    lv_style_set_radius(&style_bg, LV_RADIUS_CIRCLE);
    lv_style_set_border_width(&style_bg, 0);


    static lv_style_t style_btn;
    lv_style_init(&style_btn);
    lv_style_set_radius(&style_btn, 0);
    lv_style_set_border_width(&style_btn, 1);
    lv_style_set_border_opa(&style_btn, LV_OPA_50);
    lv_style_set_border_color(&style_btn, lv_palette_main(LV_PALETTE_GREY));
    lv_style_set_border_side(&style_btn, LV_BORDER_SIDE_INTERNAL);
    lv_style_set_radius(&style_btn, 0);

    static const char * map[] = {LV_SYMBOL_LEFT, "1", "2", "3", "4", "5", LV_SYMBOL_
↪RIGHT, ""};

    lv_obj_t * btnm = lv_btnmatrix_create(lv_scr_act());
    lv_btnmatrix_set_map(btnm, map);
    lv_obj_add_style(btnm, &style_bg, 0);
    lv_obj_add_style(btnm, &style_btn, LV_PART_ITEMS);
    lv_obj_add_event_cb(btnm, event_cb, LV_EVENT_VALUE_CHANGED, NULL);
    lv_obj_set_size(btnm, 225, 35);

    /*Allow selecting on one number at time*/
    lv_btnmatrix_set_btn_ctrl_all(btnm, LV_BTNMATRIX_CTRL_CHECKABLE);
    lv_btnmatrix_clear_btn_ctrl(btnm, 0, LV_BTNMATRIX_CTRL_CHECKABLE);
    lv_btnmatrix_clear_btn_ctrl(btnm, 6, LV_BTNMATRIX_CTRL_CHECKABLE);

    lv_btnmatrix_set_one_checked(btnm, true);
    lv_btnmatrix_set_btn_ctrl(btnm, 1, LV_BTNMATRIX_CTRL_CHECKED);

    lv_obj_center(btnm);

}

#endif
```

MicroPython

No examples yet.

API

Typedefs

typedef uint16_t **lv_btnmatrix_ctrl_t**

typedef bool (***lv_btnmatrix_btn_draw_cb_t**)(*lv_obj_t* *btnm, uint32_t btn_id, const lv_area_t *draw_area, const lv_area_t *clip_area)

Enums

enum **[anonymous]**

Type to store button control bits (disabled, hidden etc.) The first 3 bits are used to store the width

Values:

enumerator **_LV_BTNMATRIX_WIDTH**

Reserved to stire the size units

enumerator **LV_BTNMATRIX_CTRL_HIDDEN**

Button hidden

enumerator **LV_BTNMATRIX_CTRL_NO_REPEAT**

Do not repeat press this button.

enumerator **LV_BTNMATRIX_CTRL_DISABLED**

Disable this button.

enumerator **LV_BTNMATRIX_CTRL_CHECKABLE**

The button can be toggled.

enumerator **LV_BTNMATRIX_CTRL_CHECKED**

Button is currently toggled (e.g. checked).

enumerator **LV_BTNMATRIX_CTRL_CLICK_TRIG**

1: Send LV_EVENT_VALUE_CHANGE on CLICK, 0: Send LV_EVENT_VALUE_CHANGE on PRESS

enumerator **LV_BTNMATRIX_CTRL_RECOLOR**

Enable text recoloring with `#color`

enumerator **_LV_BTNMATRIX_CTRL_RESERVED**

Reserved for later use

enumerator **LV_BTNMATRIX_CTRL_CUSTOM_1**

Custom free to use flag

enumerator **LV_BTNMATRIX_CTRL_CUSTOM_2**
Custom free to use flag

Functions

LV_EXPORT_CONST_INT(LV_BTNMATRIX_BTN_NONE)

lv_obj_t ***lv_btnmatrix_create**(*lv_obj_t* *parent)
Create a button matrix objects

Parameters **parent** -- pointer to an object, it will be the parent of the new button matrix

Returns pointer to the created button matrix

void **lv_btnmatrix_set_map**(*lv_obj_t* *obj, const char *map[])
Set a new map. Buttons will be created/deleted according to the map. The button matrix keeps a reference to the map and so the string array must not be deallocated during the life of the matrix.

Parameters

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

void **lv_btnmatrix_set_ctrl_map**(*lv_obj_t* *obj, const *lv_btnmatrix_ctrl_t* ctrl_map[])
Set the button control map (hidden, disabled etc.) for a button matrix. The control map array will be copied and so may be deallocated after this function returns.

Parameters

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

void **lv_btnmatrix_set_selected_btn**(*lv_obj_t* *obj, uint16_t btn_id)
Set the selected buttons

Parameters

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

void **lv_btnmatrix_set_btn_ctrl**(*lv_obj_t* *obj, uint16_t btn_id, *lv_btnmatrix_ctrl_t* ctrl)
Set the attributes of a button of the button matrix

Parameters

- **obj** -- pointer to button matrix object
- **btn_id** -- 0 based index of the button to modify. (Not counting new lines)
- **ctrl** -- OR-ed attributs. E.g. `LV_BTNMATRIX_CTRL_NO_REPEAT | LV_BTNMATRIX_CTRL_CHECKABLE`

void **lv_btnmatrix_clear_btn_ctrl**(const *lv_obj_t* *obj, uint16_t btn_id, *lv_btnmatrix_ctrl_t* ctrl)
Clear the attributes of a button of the button matrix

Parameters

- **obj** -- pointer to button matrix object
- **btn_id** -- 0 based index of the button to modify. (Not counting new lines)
- **ctrl** -- OR-ed attributs. E.g. `LV_BTNMATRIX_CTRL_NO_REPEAT | LV_BTNMATRIX_CTRL_CHECKABLE`

void **lv_btnmatrix_set_btn_ctrl_all**(*lv_obj_t* *obj, *lv_btnmatrix_ctrl_t* ctrl)

Set attributes of all buttons of a button matrix

Parameters

- **obj** -- pointer to a button matrix object
- **ctrl** -- attribute(s) to set from `lv_btnmatrix_ctrl_t`. Values can be ORed.

void **lv_btnmatrix_clear_btn_ctrl_all**(*lv_obj_t* *obj, *lv_btnmatrix_ctrl_t* ctrl)

Clear the attributes of all buttons of a button matrix

Parameters

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

void **lv_btnmatrix_set_btn_width**(*lv_obj_t* *obj, uint16_t btn_id, uint8_t width)

Set a single button's relative width. This method will cause the matrix be regenerated and is a relatively expensive operation. It is recommended that initial width be specified using `lv_btnmatrix_set_ctrl_map` and this method only be used for dynamic changes.

Parameters

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

void **lv_btnmatrix_set_one_checked**(*lv_obj_t* *obj, bool en)

Make the button matrix like a selector widget (only one button may be checked at a time). `LV_BTNMATRIX_CTRL_CHECKABLE` must be enabled on the buttons to be selected useing `lv_btnmatrix_set_ctrl()` or `lv_btnmatrix_set_btn_ctrl_all()`.

Parameters

- **obj** -- pointer to a button matrix object
- **en** -- whether "one check" mode is enabled

const char ****lv_btnmatrix_get_map**(const *lv_obj_t* *obj)

Get the current map of a button matrix

Parameters **obj** -- pointer to a button matrix object

Returns the current map

uint16_t **lv_btnmatrix_get_selected_btn**(const *lv_obj_t* *obj)

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

Parameters **obj** -- pointer to button matrix object

Returns index of the last released button (LV_BTNMATRIX_BTN_NONE: if unset)

const char ***lv_btnmatrix_get_btn_text**(const *lv_obj_t* *obj, uint16_t btn_id)

Get the button's text

Parameters

- **obj** -- pointer to button matrix object
- **btn_id** -- the index a button not counting new line characters.

Returns text of btn_index` button

bool **lv_btnmatrix_has_btn_ctrl**(*lv_obj_t* *obj, uint16_t btn_id, *lv_btnmatrix_ctrl_t* ctrl)

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

Parameters

- **obj** -- pointer to a button matrix object
- **btn_id** -- the index of a button not counting new line characters.
- **ctrl** -- control values to check (ORed value can be used)

Returns true: the control attribute is enabled false: disabled

bool **lv_btnmatrix_get_one_checked**(const *lv_obj_t* *obj)

Tell whether "one check" mode is enabled or not.

Parameters **obj** -- Button matrix object

Returns true: "one check" mode is enabled; false: disabled

Variables

const lv_obj_class_t **lv_btnmatrix_class**

struct **lv_btnmatrix_t**

Public Members

lv_obj_t **obj**

const char ****map_p**

lv_area_t ***button_areas**

lv_btnmatrix_ctrl_t ***ctrl_bits**

uint16_t **btn_cnt**

uint16_t **btn_id_sel**

uint8_t **one_check**

5.2.5 Canvas (lv_canvas)

Overview

A Canvas inherites from *Image* where the user can draw anything. Rectangles, texts, images, lines, arcs can be drawn here using lvgl's drawing engine. Besides some "effects" can be applied as well like rotation, zoom and blur.

Parts and Styles

`LV_PART_MAIN` Uses the typical rectangle style properties and image style properties.

Usage

Buffer

The Canvas needs a buffer which stores the drawn image. To assign a buffer to a Canvas, use `lv_canvas_set_buffer(canvas, buffer, width, height, LV_IMG_CF_...)`. Where `buffer` is a static buffer (not just a local variable) to hold the image of the canvas. For example, `static lv_color_t buffer[LV_CANVAS_BUF_SIZE_TRUE_COLOR(width, height)]`. `LV_CANVAS_BUF_SIZE_...` macros help to determine the size of the buffer with different color formats.

The canvas supports all the built-in color formats like `LV_IMG_CF_TRUE_COLOR` or `LV_IMG_CF_INDEXED_2BIT`. See the full list in the Color formats section.

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

`lv_canvas_fill_bg(canvas, LV_COLOR_BLUE, LV_OPA_50)` fills the whole canvas to blue with 50% opacity. Note that, if the current color format doesn't support colors (e.g. `LV_IMG_CF_ALPHA_2BIT`) the color will be ignored. Similarly, if opacity is not supported (e.g. `LV_IMG_CF_TRUE_COLOR`) it will be ignored.

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

To draw something to the canvas use

- `lv_canvas_draw_rect(canvas, x, y, width, heigth, &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 `lv_draw_rect/label/img/line/arc_dsc_init()` function and then it's filed should be modified with the desired colors and other values.

The draw function can draw to any color format. For example, it's possible to draw a text to an `LV_IMG_VF_ALPHA_8BIT` canvas and use the result image as a *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 an other canvas too (`lv_canvas_get_img()`).
- `angle` the angle of rotation (0..3600), 0.1 deg resolution
- `zoom` zoom factor (256 no zoom, 512 double size, 128 half size);
- `offset_x` offset X to tell where to put the result data on destination canvas
- `offset_y` offset X 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 burring). `area` is the area where the blur should be applied (interpreted relative to the canvas)

Events

The same events are sent than for the *Images*.

Learn more about *Events*.

Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

Example

C

Drawing on the Canvas and rotate

code

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES


#define CANVAS_WIDTH  200
#define CANVAS_HEIGHT  150

void lv_example_canvas_1(void)
{
    lv_draw_rect_dsc_t rect_dsc;
    lv_draw_rect_dsc_init(&rect_dsc);
    rect_dsc.radius = 10;
    rect_dsc.bg_opa = LV_OPA_COVER;
    rect_dsc.bg_grad_dir = LV_GRAD_DIR_HOR;
    rect_dsc.bg_color = lv_palette_main(LV_PALETTE_RED);
    rect_dsc.bg_grad_color = lv_palette_main(LV_PALETTE_BLUE);
    rect_dsc.border_width = 2;
    rect_dsc.border_opa = LV_OPA_90;
    rect_dsc.border_color = lv_color_white();
    rect_dsc.shadow_width = 5;
    rect_dsc.shadow_ofs_x = 5;
    rect_dsc.shadow_ofs_y = 5;

    lv_draw_label_dsc_t label_dsc;
    lv_draw_label_dsc_init(&label_dsc);
    label_dsc.color = lv_palette_main(LV_PALETTE_YELLOW);

    static lv_color_t cbuf[LV_CANVAS_BUF_SIZE_TRUE_COLOR(CANVAS_WIDTH, CANVAS_
↪HEIGHT)];

    lv_obj_t * canvas = lv_canvas_create(lv_scr_act());
    lv_canvas_set_buffer(canvas, cbuf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_IMG_CF_TRUE_
↪COLOR);
    lv_obj_center(canvas);
    lv_canvas_fill_bg(canvas, lv_palette_lighten(LV_PALETTE_GREY, 3), LV_OPA_COVER);

    lv_canvas_draw_rect(canvas, 70, 60, 100, 70, &rect_dsc);

    lv_canvas_draw_text(canvas, 40, 20, 100, &label_dsc, "Some text on text canvas");

    /*Test the rotation. It requires an other buffer where the orignal image is␣
↪stored.
```

(continues on next page)

(continued from previous page)

```
     *So copy the current image to buffer and rotate it to the canvas*/
    static lv_color_t cbuf_tmp[CANVAS_WIDTH * CANVAS_HEIGHT];
    memcpy(cbuf_tmp, cbuf, sizeof(cbuf_tmp));
    lv_img_dsc_t img;
    img.data = (void *)cbuf_tmp;
    img.header.cf = LV_IMG_CF_TRUE_COLOR;
    img.header.w = CANVAS_WIDTH;
    img.header.h = CANVAS_HEIGHT;

    lv_canvas_fill_bg(canvas, lv_palette_lighten(LV_PALETTE_GREY, 3), LV_OPA_COVER);
    lv_canvas_transform(canvas, &img, 30, LV_IMG_ZOOM_NONE, 0, 0, CANVAS_WIDTH / 2,␣
↪CANVAS_HEIGHT / 2, true);
}

#endif
```

Transparent Canvas with chroma keying

code

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  50
#define CANVAS_HEIGHT  50

/**
 * Create a transparent canvas with Chroma keying and indexed color format (palette).
 */
void lv_example_canvas_2(void)
{
    /*Create a button to better see the transparency*/
    lv_btn_create(lv_scr_act());

    /*Create a buffer for the canvas*/
    static lv_color_t cbuf[LV_CANVAS_BUF_SIZE_INDEXED_1BIT(CANVAS_WIDTH, CANVAS_
↪HEIGHT)];

    /*Create a canvas and initialize its the palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_scr_act());
    lv_canvas_set_buffer(canvas, cbuf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_IMG_CF_INDEXED_
↪1BIT);
    lv_canvas_set_palette(canvas, 0, LV_COLOR_CHROMA_KEY);
    lv_canvas_set_palette(canvas, 1, lv_palette_main(LV_PALETTE_RED));

    /*Create colors with the indices of the palette*/
    lv_color_t c0;
    lv_color_t c1;

    c0.full = 0;
    c1.full = 1;

    /*Red background (There is no dedicated alpha channel in indexed images so LV_OPA_
↪COVER is ignored)*/
    lv_canvas_fill_bg(canvas, c1, LV_OPA_COVER);
```

(continues on next page)

(continued from previous page)

```
    /*Create hole on the canvas*/
    uint32_t x;
    uint32_t y;
    for( y = 10; y < 30; y++) {
        for( x = 5; x < 20; x++) {
            lv_canvas_set_px(canvas, x, y, c0);
        }
    }

}
#endif
```

MicroPython

No examples yet.

API

Functions

lv_obj_t ***lv_canvas_create**(*lv_obj_t* *parent)

Create a canvas object

Parameters **parent** -- pointer to an object, it will be the parent of the new canvas

Returns pointer to the created canvas

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

Set a buffer for the canvas.

Parameters

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

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

Set the color of a pixel on the canvas

Parameters

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

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

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

Parameters

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

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

Get the color of a pixel on the canvas

Parameters

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

Returns color of the point

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

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

Parameters **canvas** -- pointer to a canvas object

Returns pointer to the image descriptor.

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

Copy a buffer to the canvas

Parameters

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

void **lv_canvas_transform**(*lv_obj_t* *canvas, *lv_img_dsc_t* *img, int16_t angle, uint16_t zoom, lv_coord_t offset_x, lv_coord_t offset_y, int32_t pivot_x, int32_t pivot_y, bool antialias)

Transform and image and store the result on a canvas.

Parameters

- **canvas** -- pointer to a canvas object to store the result of the transformation.
- **img** -- pointer to an image descriptor to transform. Can be the image descriptor of an other canvas too (*`lv_canvas_get_img()`*).

- **angle** -- the angle of rotation (0..3600), 0.1 deg resolution
- **zoom** -- zoom factor (256 no zoom);
- **offset_x** -- offset X to tell where to put the result data on destination canvas
- **offset_y** -- offset X 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** -- apply anti-aliasing during the transformation. Looks better but slower.

void **lv_canvas_blur_hor**(*lv_obj_t* *canvas, const lv_area_t *area, uint16_t r)

Apply horizontal blur on the canvas

Parameters

- **canvas** -- pointer to a canvas object
- **area** -- the area to blur. If `NULL` the whole canvas will be blurred.
- **r** -- radius of the blur

void **lv_canvas_blur_ver**(*lv_obj_t* *canvas, const lv_area_t *area, uint16_t r)

Apply vertical blur on the canvas

Parameters

- **canvas** -- pointer to a canvas object
- **area** -- the area to blur. If `NULL` the whole canvas will be blurred.
- **r** -- radius of the blur

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

Fill the canvas with color

Parameters

- **canvas** -- pointer to a canvas
- **color** -- the background color
- **opa** -- the desired opacity

void **lv_canvas_draw_rect**(*lv_obj_t* *canvas, lv_coord_t x, lv_coord_t y, lv_coord_t w, lv_coord_t h, const lv_draw_rect_dsc_t *draw_dsc)

Draw a rectangle on the canvas

Parameters

- **canvas** -- pointer to a canvas object
- **x** -- left coordinate of the rectangle
- **y** -- top coordinate of the rectangle
- **w** -- width of the rectangle
- **h** -- height of the rectangle
- **draw_dsc** -- descriptor of the rectangle

void **lv_canvas_draw_text**(*lv_obj_t* *canvas, lv_coord_t x, lv_coord_t y, lv_coord_t max_w, lv_draw_label_dsc_t *draw_dsc, const char *txt)

Draw a text on the canvas.

Parameters

- **canvas** -- pointer to a canvas object
- **x** -- left coordinate of the text
- **y** -- top coordinate of the text
- **max_w** -- max width of the text. The text will be wrapped to fit into this size
- **draw_dsc** -- pointer to a valid label descriptor `lv_draw_label_dsc_t`
- **txt** -- text to display

void **lv_canvas_draw_img**(*lv_obj_t* *canvas, lv_coord_t x, lv_coord_t y, const void *src, const lv_draw_img_dsc_t *draw_dsc)

Draw an image on the canvas

Parameters

- **canvas** -- pointer to a canvas object
- **x** -- left coordinate of the image
- **y** -- top coordinate of the image
- **src** -- image source. Can be a pointer an *`lv_img_dsc_t`* variable or a path an image.
- **draw_dsc** -- pointer to a valid label descriptor `lv_draw_img_dsc_t`

void **lv_canvas_draw_line**(*lv_obj_t* *canvas, const lv_point_t points[], uint32_t point_cnt, const lv_draw_line_dsc_t *draw_dsc)

Draw a line on the canvas

Parameters

- **canvas** -- pointer to a canvas object
- **points** -- point of the line
- **point_cnt** -- number of points
- **draw_dsc** -- pointer to an initialized `lv_draw_line_dsc_t` variable

void **lv_canvas_draw_polygon**(*lv_obj_t* *canvas, const lv_point_t points[], uint32_t point_cnt, const lv_draw_rect_dsc_t *draw_dsc)

Draw a polygon on the canvas

Parameters

- **canvas** -- pointer to a canvas object
- **points** -- point of the polygon
- **point_cnt** -- number of points
- **draw_dsc** -- pointer to an initialized `lv_draw_rect_dsc_t` variable

void **lv_canvas_draw_arc**(*lv_obj_t* *canvas, lv_coord_t x, lv_coord_t y, lv_coord_t r, int32_t start_angle, int32_t end_angle, const lv_draw_arc_dsc_t *draw_dsc)

Draw an arc on the canvas

Parameters

- **canvas** -- pointer to a canvas object
- **x** -- origo x of the arc
- **y** -- origo y of the arc
- **r** -- radius of the arc
- **start_angle** -- start angle in degrees
- **end_angle** -- end angle in degrees
- **draw_dsc** -- pointer to an initialized `lv_draw_line_dsc_t` variable

struct **lv_canvas_t**

Public Members

lv_img_t **img**

lv_img_dsc_t **dsc**

5.2.6 Checkbox (lv_checkbox)

Overview

The Checkbox object is created from a "tick box" and a label. When the Chackbox 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 backround style properties. `pad_column` adjusts the spacing between the tickbox and the label
- `LV_PART_INDICATOR` The "tick box" is a square the uses all the typical backround style properties. By deafult its size is equal to the height of the main part's font. Padding properties makes the tick boy larger in the respectiev directions.

The Checkbox is added to the deafult group (if it is set).

Usage

Text

The text can be modified by the `lv_checkbox_set_text(cb, "New text")` function. It will dynamically allocate the text.

To set a static text, use `lv_checkbox_set_static_text(cb, txt)`. This way, only a pointer of `txt` will be stored and it shouldn't be deallocated while the checkbox exists.

Check, uncheck, disable

You can manually check, un-check, and disable the Checkbox by using the common state 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 both main and indicator parts to allow hooking the drawing. The for more detail on the main part see the Base object's documentation. For the indicator the following fields are used: `clip_area`, `draw_area`, `rect_dsc`, `part`.

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

C

Simple Checkboxes

code

```
#include "../../lv_examples.h"
#if LV_USE_CHECKBOX && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        const char * txt = lv_checkbox_get_text(obj);
        const char * state = lv_obj_get_state(obj) & LV_STATE_CHECKED ? "Checked" :
↪"Unchecked";
        LV_LOG_USER("%s: %s", txt, state);
    }
```

(continues on next page)

(continued from previous page)

```
}

void lv_example_checkbox_1(void)
{
    lv_obj_set_flex_flow(lv_scr_act(), LV_FLEX_FLOW_COLUMN);
    lv_obj_set_flex_align(lv_scr_act(), LV_FLEX_ALIGN_CENTER, LV_FLEX_ALIGN_START, LV_
↪FLEX_ALIGN_CENTER);

    lv_obj_t * cb;
    cb = lv_checkbox_create(lv_scr_act());
    lv_checkbox_set_text(cb, "Apple");
    lv_obj_add_event_cb(cb, event_handler, LV_EVENT_ALL, NULL);

    cb = lv_checkbox_create(lv_scr_act());
    lv_checkbox_set_text(cb, "Banana");
    lv_obj_add_state(cb, LV_STATE_CHECKED);
    lv_obj_add_event_cb(cb, event_handler, LV_EVENT_ALL, NULL);

    cb = lv_checkbox_create(lv_scr_act());
    lv_checkbox_set_text(cb, "Lemon");
    lv_obj_add_state(cb, LV_STATE_DISABLED);
    lv_obj_add_event_cb(cb, event_handler, LV_EVENT_ALL, NULL);

    cb = lv_checkbox_create(lv_scr_act());
    lv_obj_add_state(cb, LV_STATE_CHECKED | LV_STATE_DISABLED);
    lv_checkbox_set_text(cb, "Melon\nand a new line");
    lv_obj_add_event_cb(cb, event_handler, LV_EVENT_ALL, NULL);

    lv_obj_update_layout(cb);
}

#endif
```

MicroPython

No examples yet.

API

Functions

lv_obj_t ***lv_checkbox_create**(*lv_obj_t* *parent)

Create a check box object

Parameters **parent** -- pointer to an object, it will be the parent of the new button

Returns pointer to the created check box

void **lv_checkbox_set_text**(*lv_obj_t* *obj, const char *txt)

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

Parameters

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

void **lv_checkbox_set_text_static**(*lv_obj_t* *obj, const char *txt)

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

Parameters

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

const char ***lv_checkbox_get_text**(const *lv_obj_t* *obj)

Get the text of a check box

Parameters **cb** -- pointer to check box object

Returns pointer to the text of the check box

Variables

const lv_obj_class_t **lv_checkbox_class**

struct **lv_checkbox_t**

Public Members

lv_obj_t **obj**

char ***txt**

uint32_t **static_txt**

5.2.7 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.

The Drop-down list is added to the deafult 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 widgets is built from the elements: a "button" and a "list" (they are not realted to the butto and list widgets)

Button

- `LV_PART_MAIN` The background of the button. It uses the typicaly background proeprties and text proeprties 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 its opened.

List

- `LV_PART_MAIN` The list itself and it uses the typical background proeprties. `max_height` can be used to limit the height of the list.
- `LV_PART_SCROLLBAR` The scrollbar the background, border, shadow properties and width (for its width) and right padding for the spacing on the right.
- `LV_PART_SELECTED` Refers to the currently pressed, checked or prssed+checked option. It also uses the typical background properties.

As the list not exists when the drop-down list is closed it's not possible to simply add styles to it. Instead the following should be done:

1. Ad an event handler to the button for `LV_EVENT_VALUE_CHANGED` (triggered when the list is opened/closed)
2. Use `lv_obj_t * list = lv_dropdown_get_list(dropdown)`
3. `if(list != NULL) {/*Add the styles to the list*/}`

Alternatively the the theme can be extended with the new styles.

Usage

Overview

Set options

The options are passed to the drop-down list as a string with `lv_dropdown_set_options(dropdown, options)`. The options should be separated by `\n`. For example: `"First\nSecond\nThird"`. The string will be saved in the drop-down list, so it can in local variable too.

The `lv_dropdown_add_option(dropdown, "New option", pos)` function inserts a new option to `pos` index.

To save memory the options can 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 currently selected option, use `lv_dropdown_get_selected(dropdown)`. It will return the *index* of the selected option.

`lv_dropdown_get_selected_str(dropdown, buf, buf_size)` copies the name of the selected option to a `buf`.

Direction

The list can be created on any side. The default `LV_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 aligned to the edge.

Symbol

A symbol (typically an arrow) can be added to the drop down list with `lv_dropdown_set_symbol(dropdown, LV_SYMBOL_...)`

If the direction of the drop-down list is `LV_DIR_LEFT` the symbol will be shown on the left, else 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 th selected option. Id the text 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

Besides the Generic events, the following Special events are sent by the drop-down list:

- `LV_EVENT_VALUE_CHANGED` Sent when the new option is selected or the list is opened/closed.

Learn more about *Events*.

Keys

- `LV_KEY_RIGHT/DOWN` Select the next option.
- `LV_KEY_LEFT/UP` Select the previous option.
- `LY_KEY_ENTER` Apply the selected option (Send `LV_EVENT_VALUE_CHANGED` event and close the drop-down list).

Learn more about *Keys*.

Example

C

Simple Drop down list

code

```
#include "../../lv_examples.h"
#if LV_USE_DROPDOWN && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        char buf[32];
        lv_dropdown_get_selected_str(obj, buf, sizeof(buf));
        LV_LOG_USER("Option: %s", buf);
    }
}

void lv_example_dropdown_1(void)
{

    /*Create a normal drop down list*/
    lv_obj_t * dd = lv_dropdown_create(lv_scr_act());
    lv_dropdown_set_options(dd, "Apple\n"
                                "Banana\n"
                                "Orange\n"
                                "Cherry\n"
                                "Grape\n"
                                "Raspberry\n"
                                "Melon\n"
                                "Orange\n"
                                "Lemon\n"
                                "Nuts");

    lv_obj_align(dd, LV_ALIGN_TOP_MID, 0, 20);
    lv_obj_add_event_cb(dd, event_handler, LV_EVENT_ALL, NULL);
}

#endif
```

Drop down in four directions

code

```
#include "../../lv_examples.h"
#if LV_USE_DROPDOWN && LV_BUILD_EXAMPLES


/**
 * Create a drop down, up, left and right menus
 */
```

(continues on next page)

(continued from previous page)

```
void lv_example_dropdown_2(void)
{
    static const char * opts = "Apple\n"
                               "Banana\n"
                               "Orange\n"
                               "Melon\n"
                               "Grape\n"
                               "Raspberry";

    lv_obj_t * dd;
    dd = lv_dropdown_create(lv_scr_act());
    lv_dropdown_set_options_static(dd, opts);
    lv_obj_align(dd, LV_ALIGN_TOP_MID, 0, 10);

    dd = lv_dropdown_create(lv_scr_act());
    lv_dropdown_set_options_static(dd, opts);
    lv_dropdown_set_dir(dd, LV_DIR_BOTTOM);
    lv_dropdown_set_symbol(dd, LV_SYMBOL_UP);
    lv_obj_align(dd, LV_ALIGN_BOTTOM_MID, 0, -10);

    dd = lv_dropdown_create(lv_scr_act());
    lv_dropdown_set_options_static(dd, opts);
    lv_dropdown_set_dir(dd, LV_DIR_RIGHT);
    lv_dropdown_set_symbol(dd, LV_SYMBOL_RIGHT);
    lv_obj_align(dd, LV_ALIGN_LEFT_MID, 10, 0);

    dd = lv_dropdown_create(lv_scr_act());
    lv_dropdown_set_options_static(dd, opts);
    lv_dropdown_set_dir(dd, LV_DIR_LEFT);
    lv_dropdown_set_symbol(dd, LV_SYMBOL_LEFT);
    lv_obj_align(dd, LV_ALIGN_RIGHT_MID, -10, 0);
}

#endif
```

Menu

code

```
#include "../../lv_examples.h"
#if LV_USE_DROPDOWN && LV_BUILD_EXAMPLES

static void event_cb(lv_event_t * e)
{
    lv_obj_t * dropdown = lv_event_get_target(e);
    char buf[64];
    lv_dropdown_get_selected_str(dropdown, buf, sizeof(buf));
    LV_LOG_USER("'%s' is selected", buf);
}

/**
 * Create a menu from a drop-down list and show some drop-down list features and␣
↪styling
 */
void lv_example_dropdown_3(void)
```

(continues on next page)

(continued from previous page)

```
{
    /*Create a drop down list*/
    lv_obj_t * dropdown = lv_dropdown_create(lv_scr_act());
    lv_obj_align(dropdown, LV_ALIGN_TOP_LEFT, 10, 10);
    lv_dropdown_set_options(dropdown, "New project\n"
                                      "New file\n"
                                      "Open project\n"
                                      "Recent projects\n"
                                      "Preferences\n"
                                      "Exit");

    /*Set a fixed text to display on the button of the drop-down list*/
    lv_dropdown_set_text(dropdown, "Menu");

    /*Use a custom image as down icon and flip it when the list is opened*/
    LV_IMG_DECLARE(img_caret_down)
    lv_dropdown_set_symbol(dropdown, &img_caret_down);
    lv_obj_set_style_transform_angle(dropdown, 1800, LV_PART_INDICATOR | LV_STATE_
↪CHECKED);

    /*In a menu we don't need to show the last clicked item*/
    lv_dropdown_set_selected_highlight(dropdown, false);

    lv_obj_add_event_cb(dropdown, event_cb, LV_EVENT_VALUE_CHANGED, NULL);
}

#endif
```

MicroPython

No examples yet.

API

Functions

LV_EXPORT_CONST_INT(LV_DROPDOWN_POS_LAST)

lv_obj_t ***lv_dropdown_create**(*lv_obj_t* *parent)

Create a drop-down list objects

Parameters **parent** -- pointer to an object, it will be the parent of the new drop-down list

Returns pointer to the created drop-down list

void **lv_dropdown_set_text**(*lv_obj_t* *obj, const char *txt)

Set text of the drop-down list's button. If set to `NULL` the selected option's text will be displayed on the button. If set to a specific text then that text will be shown regardless the selected option.

Parameters

- **obj** -- pointer to a drop-down list object
- **txt** -- the text as a string (Only it's pointer is saved)

void **lv_dropdown_set_options**(*lv_obj_t* *obj, const char *options)

Set the options in a drop-down list from a string. The options will be copied and saved in the object so the `options` can be destroyed after calling this function

Parameters

- **obj** -- pointer to drop-down list object
- **options** -- a string with '

  ' separated options. E.g. "One\nTwo\nThree"

void **lv_dropdown_set_options_static**(*lv_obj_t* *obj, const char *options)

Set the options in a drop-down list from a static string (global, static or dynamically allocated). Only the pointer of the option string will be saved.

Parameters

- **obj** -- pointer to drop-down list object
- **options** -- a static string with '

  ' separated options. E.g. "One\nTwo\nThree"

void **lv_dropdown_add_option**(*lv_obj_t* *obj, const char *option, uint32_t pos)

Add an options to a drop-down list from a string. Only works for non-static options.

Parameters

- **obj** -- pointer to drop-down list object
- **option** -- a string without '

  '. E.g. "Four"

- **pos** -- the insert position, indexed from 0, LV_DROPDOWN_POS_LAST = end of string

void **lv_dropdown_clear_options**(*lv_obj_t* *obj)

Clear all options in a drop-down list. Works with both static and dynamic optins.

Parameters **obj** -- pointer to drop-down list object

void **lv_dropdown_set_selected**(*lv_obj_t* *obj, uint16_t sel_opt)

Set the selected option

Parameters

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

void **lv_dropdown_set_dir**(*lv_obj_t* *obj, lv_dir_t dir)

Set the direction of the a drop-down list

Parameters

- **obj** -- pointer to a drop-down list object
- **dir** -- LV_DIR_LEFT/RIGHT/TOP/BOTTOM

void **lv_dropdown_set_symbol**(*lv_obj_t* *obj, const void *symbol)

Set an arrow or other symbol to display when on drop-down list's button. Typically a down caret or arrow.

Note: angle and zoom transformation can be applied if the symbol is an image. E.g. when drop down is checked (opened) rotate the symbol by 180 degree

---

Parameters

- **obj** -- pointer to drop-down list object
- **symbol** -- a text like `LV_SYMBOL_DOWN`, an image (pointer or path) or NULL to not draw symbol icon

void **lv_dropdown_set_selected_highlight**(*lv_obj_t* *obj, bool en)
Set whether the selected option in the list should be highlighted or not

Parameters

- **obj** -- pointer to drop-down list object
- **en** -- true: highlight enabled; false: disabled

lv_obj_t ***lv_dropdown_get_list**(*lv_obj_t* *obj)
Get the list of a drop-down to allow styling or other modifications

Parameters **obj** -- pointer to a drop-down list object

Returns pointer to the list of the drop-down

const char ***lv_dropdown_get_text**(*lv_obj_t* *obj)
Get text of the drop-down list's button.

Parameters **obj** -- pointer to a drop-down list object

Returns the text as string, `NULL` if no text

const char ***lv_dropdown_get_options**(const *lv_obj_t* *obj)
Get the options of a drop-down list

Parameters **obj** -- pointer to drop-down list object

Returns

the options separated by '

'-s (E.g. "Option1\nOption2\nOption3")

uint16_t **lv_dropdown_get_selected**(const *lv_obj_t* *obj)
Get the index of the selected option

Parameters **obj** -- pointer to drop-down list object

Returns index of the selected option (0 ... number of option - 1);

uint16_t **lv_dropdown_get_option_cnt**(const *lv_obj_t* *obj)
Get the total number of options

Parameters **obj** -- pointer to drop-down list object

Returns the total number of options in the list

void **lv_dropdown_get_selected_str**(const *lv_obj_t* *obj, char *buf, uint32_t buf_size)
Get the current selected option as a string

Parameters

- **obj** -- pointer to drop-down object
- **buf** -- pointer to an array to store the string
- **buf_size** -- size of `buf` in bytes. 0: to ignore it.

const char ***lv_dropdown_get_symbol**(*lv_obj_t* *obj)
Get the symbol on the drop-down list. Typically a down caret or arrow.

Parameters **obj** -- pointer to drop-down list object

Returns the symbol or NULL if not enabled

bool **lv_dropdown_get_selected_highlight**(*lv_obj_t* *obj)

Get whether the selected option in the list should be highlighted or not

Parameters **obj** -- pointer to drop-down list object

Returns true: highlight enabled; false: disabled

lv_dir_t **lv_dropdown_get_dir**(const *lv_obj_t* *obj)

Get the direction of the drop-down list

Parameters **obj** -- pointer to a drop-down list object

Returns LV_DIR_LEF/RIGHT/TOP/BOTTOM

void **lv_dropdown_open**(*lv_obj_t* *dropdown_obj)

Open the drop.down list

Parameters **obj** -- pointer to drop-down list object

void **lv_dropdown_close**(*lv_obj_t* *obj)

Close (Collapse) the drop-down list

Parameters **obj** -- pointer to drop-down list object

Variables

const lv_obj_class_t **lv_dropdown_class**

const lv_obj_class_t **lv_dropdownlist_class**

struct **lv_dropdown_t**

Public Members

lv_obj_t **obj**

lv_obj_t ***list**

The dropped down list

const char ***text**

Text to display on the dropdown's button

const void ***symbol**

Arrow or other icon when the drop-down list is closed

char ***options**

Options in a a '

' separated list

uint16_t **option_cnt**

Number of options

uint16_t **sel_opt_id**
Index of the currently selected option

uint16_t **sel_opt_id_orig**
Store the original index on focus

uint16_t **pr_opt_id**
Index of the currently pressed option

lv_dir_t **dir**
Direction in which the list should open

uint8_t **static_txt**
1: Only a pointer is saved in `options`

uint8_t **selected_highlight**
1: Make the selected option highlighted in the list

struct **lv_dropdown_list_t**

Public Members

lv_obj_t **obj**

lv_obj_t ***dropdown**

5.2.8 Image (lv_img)

Overview

Images are the basic object to display images from the flash (as arrays) or externally as files. Images can display symbols (`LV_SYMBOL_...`) too.

Using the Image decoder interface custom image formats can be supported as well.

Parts and Styles

- `LV_PART_MAIN` A background rectangle that uses the typical background style proeprties and the image itself using teh image style proeprties.

Usage

Image source

To provide maximum flexibility, the source of the image can be:

- a variable in the code (a C array with the pixels).
- a file stored externally (like on an SD card).
- a text with *Symbols*.

To set the source of an image, use `lv_img_set_src(img, src)`.

To generate a pixel array from a PNG, JPG or BMP image, use the Online image converter tool and set the converted image with its pointer: `lv_img_set_src(img1, &converted_img_var);` To make the variable visible in the C file, you need to declare it with `LV_IMG_DECLARE(converted_img_var)`.

To use external files, you also need to convert the image files using the online converter tool but now you should select the binary output format. You also need to use LVGL's file system module and register a driver with some functions for the basic file operation. Go to the *File system* to learn more. To set an image sourced from a file, use `lv_img_set_src(img, "S:folder1/my_img.bin")`.

You can set a symbol similarly to *Labels*. In this case, the image will be rendered as text according to the *font* specified in the style. It enables to use of light-weighted mono-color "letters" instead of real images. You can set symbol like `lv_img_set_src(img1, LV_SYMBOL_OK)`.

Label as an image

Images and labels are sometimes used to convey the same thing. For example, to describe what a button does. Therefore, images and labels are somewhat interchangeable, 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:

- **Chrome 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 *True color* (RGB) color format, the following formats are also supported:

- **Indexed** - Image has a palette.
- **Alpha indexed** - Only alpha values are stored.

These options can be selected in the image converter. To learn more about the color formats, read the *Images* section.

Recolor

A color can be mixed to every pixel of an image with a given intensity. It is very 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

Is the width or height of the image object is set to `LV_SIZE_CONTENT` the obejct's size will be set according to the size of 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. It's a very useful feature to create a large image from only a very narrow source. For example, you can have a *300 x 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. It is useful if the object size is smaller than the image source size. Using the offset parameter a Texture atlas or a "running image" effect can be created by *Animating* the x or y offset.

Transformations

Using the `lv_img_set_zoom(img, factor)` the images will be zoomed. Set `factor` to `256` or `LV_IMG_ZOOM_NONE` to disable zooming. A larger value enlarges the images (e.g. `512` double size), a smaller value shrinks it (e.g. `128` half size). Fractional scale works as well. E.g. `281` for 10% enlargement.

To rotate the image use `lv_img_set_angle(img, angle)`. Angle has 0.1 degree precision, so for 45.8° set 458.

The `transform_zoom` and `transform_angle` style proeprties are also used to determin 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 has a higher quality but they are slower.

The transformations require the whole image to be available. Therefore indexed images (`LV_IMG_CF_INDEXED_...`), alpha only images (`LV_IMG_CF_ALPHA_...`) or images from files can 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 object won't change during transformation. That is `lv_obj_get_width/height/x/y()` will returned the original, non-zoomed coordinates.

Events

No special events are sendt by the imge objects.

Learn more about *Events*.

Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

Example

C

Image from variable and symbol

code

```
#include "../../lv_examples.h"
#if LV_USE_IMG && LV_BUILD_EXAMPLES


void lv_example_img_1(void)
{
    LV_IMG_DECLARE(img_cogwheel_argb);
    lv_obj_t * img1 = lv_img_create(lv_scr_act());
    lv_img_set_src(img1, &img_cogwheel_argb);
    lv_obj_align(img1, LV_ALIGN_CENTER, 0, -20);
    lv_obj_set_size(img1, 200, 200);

    lv_obj_t * img2 = lv_img_create(lv_scr_act());
    lv_img_set_src(img2, LV_SYMBOL_OK "Accept");
    lv_obj_align_to(img2, img1, LV_ALIGN_OUT_BOTTOM_MID, 0, 20);
}

#endif
```

Image recoloring

code

```
#include "../../lv_examples.h"
#if LV_USE_IMG && LV_USE_SLIDER && LV_BUILD_EXAMPLES

static lv_obj_t * create_slider(lv_color_t color);
static void slider_event_cb(lv_event_t * e);

static lv_obj_t * red_slider, * green_slider, * blue_slider, * intense_slider;
static lv_obj_t * img1;
```

(continues on next page)

(continued from previous page)

```
/**
 * Demonstrate runtime image re-coloring
 */
void lv_example_img_2(void)
{
    /*Create 4 sliders to adjust RGB color and re-color intensity*/
    red_slider = create_slider(lv_palette_main(LV_PALETTE_RED));
    green_slider = create_slider(lv_palette_main(LV_PALETTE_GREEN));
    blue_slider = create_slider(lv_palette_main(LV_PALETTE_BLUE));
    intense_slider = create_slider(lv_palette_main(LV_PALETTE_GREY));

    lv_slider_set_value(red_slider, LV_OPA_20, LV_ANIM_OFF);
    lv_slider_set_value(green_slider, LV_OPA_90, LV_ANIM_OFF);
    lv_slider_set_value(blue_slider, LV_OPA_60, LV_ANIM_OFF);
    lv_slider_set_value(intense_slider, LV_OPA_50, LV_ANIM_OFF);

    lv_obj_align(red_slider, LV_ALIGN_LEFT_MID, 25, 0);
    lv_obj_align_to(green_slider, red_slider, LV_ALIGN_OUT_RIGHT_MID, 25, 0);
    lv_obj_align_to(blue_slider, green_slider, LV_ALIGN_OUT_RIGHT_MID, 25, 0);
    lv_obj_align_to(intense_slider, blue_slider, LV_ALIGN_OUT_RIGHT_MID, 25, 0);

    /*Now create the actual image*/
    LV_IMG_DECLARE(img_cogwheel_argb)
    img1 = lv_img_create(lv_scr_act());
    lv_img_set_src(img1, &img_cogwheel_argb);
    lv_obj_align(img1, LV_ALIGN_RIGHT_MID, -20, 0);

    lv_event_send(intense_slider, LV_EVENT_VALUE_CHANGED, NULL);
}

static void slider_event_cb(lv_event_t * e)
{
    LV_UNUSED(e);

    /*Recolor the image based on the sliders' values*/
    lv_color_t color  = lv_color_make(lv_slider_get_value(red_slider), lv_slider_get_
↪value(green_slider), lv_slider_get_value(blue_slider));
    lv_opa_t intense = lv_slider_get_value(intense_slider);
    lv_obj_set_style_img_recolor_opa(img1, intense, 0);
    lv_obj_set_style_img_recolor(img1, color, 0);
}

static lv_obj_t * create_slider(lv_color_t color)
{
    lv_obj_t * slider = lv_slider_create(lv_scr_act());
    lv_slider_set_range(slider, 0, 255);
    lv_obj_set_size(slider, 10, 200);
    lv_obj_set_style_bg_color(slider, color, LV_PART_KNOB);
    lv_obj_set_style_bg_color(slider, lv_color_darken(color, LV_OPA_40), LV_PART_
↪INDICATOR);
    lv_obj_add_event_cb(slider, slider_event_cb, LV_EVENT_VALUE_CHANGED, NULL);
    return slider;
}

#endif
```

Rotate and zoom

code

```
#include "../../lv_examples.h"
#if LV_USE_IMG && LV_BUILD_EXAMPLES

static void set_angle(void * img, int32_t v)
{
    lv_img_set_angle(img, v);
}

static void set_zoom(void * img, int32_t v)
{
    lv_img_set_zoom(img, v);
}


/**
 * Show transformations (zoom and rotation) using a pivot point.
 */
void lv_example_img_3(void)
{
    LV_IMG_DECLARE(img_cogwheel_argb);

    /*Now create the actual image*/
    lv_obj_t * img = lv_img_create(lv_scr_act());
    lv_img_set_src(img, &img_cogwheel_argb);
    lv_obj_align(img, LV_ALIGN_CENTER, 50, 50);
    lv_img_set_pivot(img, 0, 0);    /*Rotate around the top left corner*/

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, img);
    lv_anim_set_exec_cb(&a, set_angle);
    lv_anim_set_values(&a, 0, 3600);
    lv_anim_set_time(&a, 5000);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);
    lv_anim_start(&a);

    lv_anim_set_exec_cb(&a, set_zoom);
    lv_anim_set_values(&a, 128, 256);
    lv_anim_set_playback_time(&a, 3000);
    lv_anim_start(&a);

}

#endif
```

Image offset and styling

code

```
#include "../../lv_examples.h"
#if LV_USE_IMG && LV_BUILD_EXAMPLES

static void ofs_y_anim(void * img, int32_t v)
{
    lv_img_set_offset_y(img, v);
}

/**
 * Image styling and offset
 */
void lv_example_img_4(void)
{
    LV_IMG_DECLARE(img_skew_strip);

    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_bg_color(&style, lv_palette_main(LV_PALETTE_YELLOW));
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_img_recolor_opa(&style, LV_OPA_COVER);
    lv_style_set_img_recolor(&style, lv_color_black());

    lv_obj_t * img = lv_img_create(lv_scr_act());
    lv_obj_add_style(img, &style, 0);
    lv_img_set_src(img, &img_skew_strip);
    lv_obj_set_size(img, 150, 100);
    lv_obj_center(img);

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, img);
    lv_anim_set_exec_cb(&a, ofs_y_anim);
    lv_anim_set_values(&a, 0, 100);
    lv_anim_set_time(&a, 3000);
    lv_anim_set_playback_time(&a, 500);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);
    lv_anim_start(&a);

}

#endif
```

MicroPython

No examples yet.

API

Functions

lv_obj_t ***lv_img_create**(*lv_obj_t* *parent)

Create a image objects

Parameters **parent** -- pointer to an object, it will be the parent of the new image

Returns pointer to the created image

void **lv_img_set_src**(*lv_obj_t* *obj, const void *src)

Set the image data to display on the the object

Parameters

- **obj** -- pointer to an image object
- **src_img** -- 1) pointer to an *lv_img_dsc_t* descriptor (converted by LVGL's image converter) (e.g. &my_img) or 2) path to an image file (e.g. "S:/dir/img.bin")or 3) a SYMBOL (e.g. LV_SYMBOL_OK)

void **lv_img_set_offset_x**(*lv_obj_t* *obj, lv_coord_t x)

Set an offset for the source of an image so the image will be displayed from the new origin.

Parameters

- **obj** -- pointer to an image
- **x** -- the new offset along x axis.

void **lv_img_set_offset_y**(*lv_obj_t* *obj, lv_coord_t y)

Set an offset for the source of an image. so the image will be displayed from the new origin.

Parameters

- **obj** -- pointer to an image
- **y** -- the new offset along y axis.

void **lv_img_set_angle**(*lv_obj_t* *obj, int16_t angle)

Set the rotation angle of the image. The image will be rotated around the set pivot set by `lv_img_set_pivot()`

Parameters

- **obj** -- pointer to an image object
- **angle** -- rotation angle in degree with 0.1 degree resolution (0..3600: clock wise)

void **lv_img_set_pivot**(*lv_obj_t* *obj, lv_coord_t x, lv_coord_t y)

Set the rotation center of the image. The image will be rotated around this point

Parameters

- **obj** -- pointer to an image object
- **x** -- rotation center x of the image
- **y** -- rotation center y of the image

void **lv_img_set_zoom**(*lv_obj_t* *obj, uint16_t zoom)

void **lv_img_set_antialias**(*lv_obj_t* *obj, bool antialias)

Enable/disable anti-aliasing for the transformations (rotate, zoom) or not. The qualitiy is better with anti-aliasing looks better but slower.

Parameters

- **obj** -- pointer to an image object
- **antialias** -- true: anti-aliased; false: not anti-aliased

const void ***lv_img_get_src**(*lv_obj_t* *obj)

Get the source of the image

Parameters **obj** -- pointer to an image object

Returns the image source (symbol, file name or ::lv-img_dsc_t for C arrays)

lv_coord_t **lv_img_get_offset_x**(*lv_obj_t* *obj)

Get the offset's x attribute of the image object.

Parameters **img** -- pointer to an image

Returns offset X value.

lv_coord_t **lv_img_get_offset_y**(*lv_obj_t* *obj)

Get the offset's y attribute of the image object.

Parameters **obj** -- pointer to an image

Returns offset Y value.

uint16_t **lv_img_get_angle**(*lv_obj_t* *obj)

Get the rotation angle of the image.

Parameters **obj** -- pointer to an image object

Returns rotation angle in 0.1 degrees (0..3600)

void **lv_img_get_pivot**(*lv_obj_t* *obj, lv_point_t *pivot)

Get the pivot (rotation center) of the image.

Parameters

- **img** -- pointer to an image object
- **pivot** -- store the rotation center here

uint16_t **lv_img_get_zoom**(*lv_obj_t* *obj)

Get the zoom factor of the image.

Parameters **obj** -- pointer to an image object

Returns zoom factor (256: no zoom)

bool **lv_img_get_antialias**(*lv_obj_t* *obj)

Get whether the transformations (rotate, zoom) are anti-aliased or not

Parameters **obj** -- pointer to an image object

Returns true: anti-aliased; false: not anti-aliased

Variables

const lv_obj_class_t **lv_img_class**

struct **lv_img_t**

Public Members

lv_obj_t **obj**

const void ***src**

lv_point_t **offset**

lv_coord_t **w**

lv_coord_t **h**

uint16_t **angle**

lv_point_t **pivot**

uint16_t **zoom**

uint8_t **src_type**

uint8_t **cf**

uint8_t **antialias**

5.2.9 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")`. It will allocate a buffer dynamically, and the provided string will be copied into that buffer. Therefore, you don't need to keep the text you pass to `lv_label_set_text` in scope after that function returns.

With `lv_label_set_text_fmt(label, "Value: %d", 15)` printf formatting can be used to set the text.

Labels are able to show text from a static character buffer. 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.

New line

New line 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 expands to the text size. Otherwise, if the width or height is explicitly set (useing 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. Similary, 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, elst 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 continously. 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 of 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 when on PC a you use your mouse to select a text. The whole mechanzim (click and select the text as you drag your finger/mouse) is implemeted in *Text area* and the Label widget allows only to manually make parts of the text selected with `lv_label_get_text_selection_start(label, start_char_index)` and `lv_label_get_text_selection_start(label, end_char_index)`.

Very long texts

LVGL can efficiently handle very long (e.g. > 40k characters) by saving some extra data (~12 bytes) to speed up drawing. To enable this feature, set `LV_LABEL_LONG_TXT_HINT 1` in `lv_conf.h`.

Symbols

The labels can display symbols alongside letters (or on their own). Read the *Font* section to learn more about the symbols.

Events

No special event's are send by the Label.

Learn more about *Events*.

Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

Example

C

Line wrap, recoloring and scrolling

code

```
#include "../../lv_examples.h"
#if LV_USE_LABEL && LV_BUILD_EXAMPLES

/**
 * Show line wrap, re-color, line align and text scrolling.
 */
```

(continues on next page)

(continued from previous page)

```
void lv_example_label_1(void)
{
    lv_obj_t * label1 = lv_label_create(lv_scr_act());
    lv_label_set_long_mode(label1, LV_LABEL_LONG_WRAP);     /*Break the long lines*/
    lv_label_set_recolor(label1, true);                      /*Enable re-coloring by␣
↪commands in the text*/
    lv_label_set_text(label1, "#0000ff Re-color# #ff00ff words# #ff0000 of a# label,␣
↪align the lines to the center"
                              "and  wrap long text automatically.");
    lv_obj_set_width(label1, 150);  /*Set smaller width to make the lines wrap*/
    lv_obj_set_style_text_align(label1, LV_TEXT_ALIGN_CENTER, 0);
    lv_obj_align(label1, LV_ALIGN_CENTER, 0, -40);


    lv_obj_t * label2 = lv_label_create(lv_scr_act());
    lv_label_set_long_mode(label2, LV_LABEL_LONG_SCROLL_CIRCULAR);     /*Circular␣
↪scroll*/
    lv_obj_set_width(label2, 150);
    lv_label_set_text(label2, "It is a circularly scrolling text. ");
    lv_obj_align(label2, LV_ALIGN_CENTER, 0, 40);
}

#endif
```

Text shadow

code

```
#include "../../lv_examples.h"
#if LV_USE_LABEL && LV_BUILD_EXAMPLES

/**
 * Create a fake text shadow
 */
void lv_example_label_2(void)
{
    /*Create a style for the shadow*/
    static lv_style_t style_shadow;
    lv_style_init(&style_shadow);
    lv_style_set_text_opa(&style_shadow, LV_OPA_30);
    lv_style_set_text_color(&style_shadow, lv_color_black());

    /*Create a label for the shadow first (it's in the background)*/
    lv_obj_t * shadow_label = lv_label_create(lv_scr_act());
    lv_obj_add_style(shadow_label, &style_shadow, 0);

    /*Create the main label*/
    lv_obj_t * main_label = lv_label_create(lv_scr_act());
    lv_label_set_text(main_label, "A simple method to create\n"
                                  "shadows on a text.\n"
                                  "It even works with\n\n"
                                  "newlines     and spaces.");

    /*Set the same text for the shadow label*/
    lv_label_set_text(shadow_label, lv_label_get_text(main_label));
```

(continues on next page)

(continued from previous page)

```
    /*Position the main label*/
    lv_obj_align(main_label, LV_ALIGN_CENTER, 0, 0);

    /*Shift the second label down and to the right by 2 pixel*/
    lv_obj_align_to(shadow_label, main_label, LV_ALIGN_TOP_LEFT, 2, 2);
}

#endif
```

MicroPython

No examples yet.

API

Typedefs

typedef uint8_t **lv_label_long_mode_t**

Enums

enum **[anonymous]**

Long mode behaviors. Used in 'lv_label_ext_t'

Values:

enumerator **LV_LABEL_LONG_WRAP**

Keep the object width, wrap the too long lines and expand the object height

enumerator **LV_LABEL_LONG_DOT**

Keep the size and write dots at the end if the text is too long

enumerator **LV_LABEL_LONG_SCROLL**

Keep the size and roll the text back and forth

enumerator **LV_LABEL_LONG_SCROLL_CIRCULAR**

Keep the size and roll the text circularly

enumerator **LV_LABEL_LONG_CLIP**

Keep the size and clip the text out of it

Functions

LV_EXPORT_CONST_INT(LV_LABEL_DOT_NUM)

LV_EXPORT_CONST_INT(LV_LABEL_POS_LAST)

LV_EXPORT_CONST_INT(LV_LABEL_TEXT_SELECTION_OFF)

lv_obj_t ***lv_label_create**(*lv_obj_t* *parent)

Create a label objects

Parameters **parent** -- pointer to an object, it will be the parent of the new labely.

Returns pointer to the created button

void **lv_label_set_text**(*lv_obj_t* *obj, const char *text)

Set a new text for a label. Memory will be allocated to store the text by the label.

Parameters

- **label** -- pointer to a label object
- **text** -- '\0' terminated character string. NULL to refresh with the current text.

void **lv_label_set_text_fmt**(*lv_obj_t* *obj, const char *fmt, ...)

void **lv_label_set_text_static**(*lv_obj_t* *obj, const char *text)

Set a static text. It will not be saved by the label so the 'text' variable has to be 'alive' while the label exist.

Parameters

- **label** -- pointer to a label object
- **text** -- pointer to a text. NULL to refresh with the current text.

void **lv_label_set_long_mode**(*lv_obj_t* *obj, *lv_label_long_mode_t* long_mode)

Set the behavior of the label with longer text then the object size

Parameters

- **label** -- pointer to a label object
- **long_mode** -- the new mode from 'lv_label_long_mode' enum. In LV_LONG_WRAP/DOT/SCROLL/SCROLL_CIRC the size of the label should be set AFTER this function

void **lv_label_set_recolor**(*lv_obj_t* *obj, bool en)

void **lv_label_set_text_sel_start**(*lv_obj_t* *obj, uint32_t index)

Set where text selection should start

Parameters

- **obj** -- pointer to a label object
- **index** -- character index from where selection should start. LV_LABEL_TEXT_SELECTION_OFF for no selection

void **lv_label_set_text_sel_end**(*lv_obj_t* *obj, uint32_t index)

Set where text selection should end

Parameters

- **obj** -- pointer to a label object
- **index** -- character index where selection should end. `LV_LABEL_TEXT_SELECTION_OFF` for no selection

char ***lv_label_get_text**(const *lv_obj_t* *obj)
Get the text of a label

Parameters **obj** -- pointer to a label object

Returns the text of the label

lv_label_long_mode_t **lv_label_get_long_mode**(const *lv_obj_t* *obj)
Get the long mode of a label

Parameters **obj** -- pointer to a label object

Returns the current long mode

bool **lv_label_get_recolor**(const *lv_obj_t* *obj)
Get the recoloring attribute

Parameters **obj** -- pointer to a label object

Returns true: recoloring is enabled, false: disable

void **lv_label_get_letter_pos**(const *lv_obj_t* *obj, uint32_t char_id, lv_point_t *pos)
Get the relative x and y coordinates of a letter

Parameters

- **obj** -- pointer to a label object
- **index** -- index of the character [0 ... text length - 1]. Expressed in character index, not byte index (different in UTF-8)
- **pos** -- store the result here (E.g. index = 0 gives 0;0 coordinates if the text if aligned to the left)

uint32_t **lv_label_get_letter_on**(const *lv_obj_t* *obj, lv_point_t *pos_in)
Get the index of letter on a relative point of a label.

Parameters

- **obj** -- pointer to label object
- **pos** -- pointer to point with coordinates on a the label

Returns The index of the letter on the 'pos_p' point (E.g. on 0;0 is the 0. letter if aligned to the left) Expressed in character index and not byte index (different in UTF-8)

bool **lv_label_is_char_under_pos**(const *lv_obj_t* *obj, lv_point_t *pos)
Check if a character is drawn under a point.

Parameters

- **label** -- Label object
- **pos** -- Point to check for character under

Returns whether a character is drawn under the point

uint32_t **lv_label_get_text_selection_start**(const *lv_obj_t* *obj)
Get the selection start index.

Parameters **obj** -- pointer to a label object.

Returns selection start index. `LV_LABEL_TEXT_SELECTION_OFF` if nothing is selected.

uint32_t **`lv_label_get_text_selection_end`**(const *lv_obj_t* *obj)

Get the selection end index.

Parameters **`obj`** -- pointer to a label object.

Returns selection end index. `LV_LABEL_TXT_SEL_OFF` if nothing is selected.

void **`lv_label_ins_text`**(*lv_obj_t* *obj, uint32_t pos, const char *txt)

Insert a text to a label. The label text can not be static.

Parameters

- **`obj`** -- pointer to a label object
- **`pos`** -- character index to insert. Expressed in character index and not byte index. 0: before first char. LV_LABEL_POS_LAST: after last char.
- **`txt`** -- pointer to the text to insert

void **`lv_label_cut_text`**(*lv_obj_t* *obj, uint32_t pos, uint32_t cnt)

Delete characters from a label. The label text can not be static.

Parameters

- **`label`** -- pointer to a label object
- **`pos`** -- character index from where to cut. Expressed in character index and not byte index. 0: start in from of the first character
- **`cnt`** -- number of characters to cut

Variables

const lv_obj_class_t **`lv_label_class`**

struct **`lv_label_t`**

Public Members

lv_obj_t **`obj`**

char ***`text`**

char ***`tmp_ptr`**

char **`tmp`**[LV_LABEL_DOT_NUM + 1]

union *lv_label_t*::[anonymous] **`dot`**

uint32_t **`dot_end`**

lv_draw_label_hint_t **`hint`**

uint32_t **`sel_start`**

uint32_t **`sel_end`**

lv_point_t **`offset`**

lv_label_long_mode_t **`long_mode`**

uint8_t **static_txt**

uint8_t **recolor**

uint8_t **expand**

uint8_t **dot_tmp_alloc**

5.2.10 Line (lv_line)

Overview

The Line object is capable of drawing straight lines between a set of points.

Parts and Styles

- `LV_PART_MAIN` It uses all the typical backgrund properties and the line style properties.

Usage

Set points

The points has to be stored in an `lv_point_t` array and passed to the object by the `lv_line_set_points(lines, point_array, point_cnt)` function.

Auto-size

By default the Line's width and height is set to `LV_SIZE_CONTENT` to automatically set its size to involve all the points. If the size if set explicitly the point out of the object It can be enable with the `lv_line_set_auto_size(line, true)` function. If enabled then when the points are set the object's width and height will be changed according to the maximal x and y coordinates among the points. The *auto size* is enabled by default.

Invert y

By deafult, the *y == 0* point is in the top of the object. It might be conter-intuitive in some cases so the y coordinates can be inverted with `lv_line_set_y_invert(line, true)`. In this case, *y == 0* will be the bottom of teh obejct. The *y invert* is disabled by default.

Events

Only the Generic events are sent by the object type.

Learn more about *Events*.

Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

Example

C

Simple Line

code

```
#include "../../lv_examples.h"
#if LV_USE_LINE && LV_BUILD_EXAMPLES

void lv_example_line_1(void)
{
    /*Create an array for the points of the line*/
    static lv_point_t line_points[] = { {5, 5}, {70, 70}, {120, 10}, {180, 60}, {240,␣
↪10} };

    /*Create style*/
    static lv_style_t style_line;
    lv_style_init(&style_line);
    lv_style_set_line_width(&style_line, 8);
    lv_style_set_line_color(&style_line, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_line_rounded(&style_line, true);

    /*Create a line and apply the new style*/
    lv_obj_t * line1;
    line1 = lv_line_create(lv_scr_act());
    lv_line_set_points(line1, line_points, 5);     /*Set the points*/
    lv_obj_add_style(line1, &style_line, 0);
    lv_obj_center(line1);
}

#endif
```

MicroPython

No examples yet.

API

Functions

lv_obj_t ***lv_line_create**(*lv_obj_t* *parent)

Create a line objects

Parameters **par** -- pointer to an object, it will be the parent of the new line

Returns pointer to the created line

void **lv_line_set_points**(*lv_obj_t* *obj, const lv_point_t points[], uint16_t point_num)

Set an array of points. The line object will connect these points.

Parameters

- **obj** -- pointer to a line object
- **points** -- an array of points. Only the address is saved, so the array needs to be alive while the line exists
- **point_num** -- number of points in 'point_a'

void **lv_line_set_y_invert**(*lv_obj_t* *obj, bool en)

Enable (or disable) the y coordinate inversion. If enabled then y will be subtracted from the height of the object, therefore the y = 0 coordinate will be on the bottom.

Parameters

- **obj** -- pointer to a line object
- **en** -- true: enable the y inversion, false:disable the y inversion

bool **lv_line_get_y_invert**(const *lv_obj_t* *obj)

Get the y inversion attribute

Parameters **obj** -- pointer to a line object

Returns true: y inversion is enabled, false: disabled

Variables

const lv_obj_class_t **lv_line_class**

struct **lv_line_t**

Public Members

lv_obj_t **obj**

const lv_point_t ***point_array**
Pointer to an array with the points of the line

uint16_t **point_num**
Number of points in 'point_array'

uint8_t **y_inv**
1: y == 0 will be on the bottom

5.2.11 Roller (lv_roller)

Overview

Roller allows you to simply select one option from more with scrolling.

Parts and Styles

- `LV_PART_MAIN` The background of the roller that uses all the typical background properties and the 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 it's 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

The options are passed to the Roller as a string with `lv_roller_set_options(roller, options, LV_ROLLER_MODE_NORMAL/INFINITE)`. The options should be separated by `\n`. For example: `"First\nSecond\nThird"`.

`LV_ROLLER_MODE_INFINITE` make the roller circular.

You can select an option manually with `lv_roller_set_selected(roller, id, LV_ANIM_ON/OFF)`, where *id* is the index of an option.

Get selected option

The get the currently selected option use `lv_roller_get_selected(roller)` it will return the *index* of the selected option.

`lv_roller_get_selected_str(roller, buf, buf_size)` copy the name of the selected option to `buf`.

Visible rows

The number of visible rows can be adjusted with `lv_roller_set_visible_row_count(roller, num)`

Events

- `LV_EVENT_VALUE_CHANGED` Sent when a new option is selected.

Learn more about *Events*.

Keys

- `LV_KEY_RIGHT/DOWN` Select the next option
- `LV_KEY_LEFT/UP` Select the previous option
- `LY_KEY_ENTER` Apply the selected option (Send `LV_EVENT_VALUE_CHANGED` event)

Example

C

Simple Roller

code

```
#include "../../lv_examples.h"
#if LV_USE_ROLLER && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        char buf[32];
        lv_roller_get_selected_str(obj, buf, sizeof(buf));
        LV_LOG_USER("Selected month: %s\n", buf);
    }
}

/**
 * An infinite roller with the name of the months
 */
void lv_example_roller_1(void)
{
    lv_obj_t *roller1 = lv_roller_create(lv_scr_act());
```

(continues on next page)

(continued from previous page)

```
    lv_roller_set_options(roller1,
                          "January\n"
                          "February\n"
                          "March\n"
                          "April\n"
                          "May\n"
                          "June\n"
                          "July\n"
                          "August\n"
                          "September\n"
                          "October\n"
                          "November\n"
                          "December",
                          LV_ROLLER_MODE_INFINITE);

    lv_roller_set_visible_row_count(roller1, 4);
    lv_obj_center(roller1);
    lv_obj_add_event_cb(roller1, event_handler, LV_EVENT_ALL, NULL);
}

#endif
```

Styling the roller

code

```
#include "../../lv_examples.h"
#if LV_USE_ROLLER && LV_FONT_MONTSERRAT_22 && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        char buf[32];
        lv_roller_get_selected_str(obj, buf, sizeof(buf));
        LV_LOG_USER("Selected value: %s", buf);
    }
}

/**
 * Roller with various alignments and larger text in the selected area
 */
void lv_example_roller_2(void)
{
    /*A style to make the selected option larger*/
    static lv_style_t style_sel;
    lv_style_init(&style_sel);
    lv_style_set_text_font(&style_sel, &lv_font_montserrat_22);

    const char * opts = "1\n2\n3\n4\n5\n6\n7\n8\n9\n10";
    lv_obj_t *roller;

    /*A roller on the left with left aligned text, and custom width*/
    roller = lv_roller_create(lv_scr_act());
```

(continues on next page)

(continued from previous page)

```
    lv_roller_set_options(roller, opts, LV_ROLLER_MODE_NORMAL);
    lv_roller_set_visible_row_count(roller, 2);
    lv_obj_set_width(roller, 100);
    lv_obj_add_style(roller, &style_sel, LV_PART_SELECTED);
    lv_obj_set_style_text_align(roller, LV_TEXT_ALIGN_LEFT, 0);
    lv_obj_align(roller, LV_ALIGN_LEFT_MID, 10, 0);
    lv_obj_add_event_cb(roller, event_handler, LV_EVENT_ALL, NULL);
    lv_roller_set_selected(roller, 2, LV_ANIM_OFF);

    /*A roller on the middle with center aligned text, and auto (default) width*/
    roller = lv_roller_create(lv_scr_act());
    lv_roller_set_options(roller, opts, LV_ROLLER_MODE_NORMAL);
    lv_roller_set_visible_row_count(roller, 3);
    lv_obj_add_style(roller, &style_sel, LV_PART_SELECTED);
    lv_obj_align(roller, LV_ALIGN_CENTER, 0, 0);
    lv_obj_add_event_cb(roller, event_handler, LV_EVENT_ALL, NULL);
    lv_roller_set_selected(roller, 5, LV_ANIM_OFF);

    /*A roller on the right with right aligned text, and custom width*/
    roller = lv_roller_create(lv_scr_act());
    lv_roller_set_options(roller, opts, LV_ROLLER_MODE_NORMAL);
    lv_roller_set_visible_row_count(roller, 4);
    lv_obj_set_width(roller, 80);
    lv_obj_add_style(roller, &style_sel, LV_PART_SELECTED);
    lv_obj_set_style_text_align(roller, LV_TEXT_ALIGN_RIGHT, 0);
    lv_obj_align(roller, LV_ALIGN_RIGHT_MID, -10, 0);
    lv_obj_add_event_cb(roller, event_handler, LV_EVENT_ALL, NULL);
    lv_roller_set_selected(roller, 8, LV_ANIM_OFF);
}

#endif
```

MicroPython

No examples yet.

API

Typedefs

typedef uint8_t **`lv_roller_mode_t`**

Enums

enum **[anonymous]**
Roller mode.

Values:

enumerator **LV_ROLLER_MODE_NORMAL**
Normal mode (roller ends at the end of the options).

enumerator **LV_ROLLER_MODE_INFINITE**
Infinite mode (roller can be scrolled forever).

Functions

lv_obj_t ***lv_roller_create**(*lv_obj_t* *parent)
Create a roller objects

Parameters **parent** -- pointer to an object, it will be the parent of the new roller.

Returns pointer to the created roller

void **lv_roller_set_options**(*lv_obj_t* *obj, const char *options, *lv_roller_mode_t* mode)
Set the options on a roller

Parameters

- **obj** -- pointer to roller object
- **options** -- a string with '

  ' separated options. E.g. "One\nTwo\nThree"
- **mode** -- `LV_ROLLER_MODE_NORMAL` or `LV_ROLLER_MODE_INFINITE`

void **lv_roller_set_selected**(*lv_obj_t* *obj, uint16_t sel_opt, *lv_anim_enable_t* anim)
Set the selected option

Parameters

- **obj** -- pointer to a roller object
- **sel_opt** -- index of the selected option (0 ... number of option - 1);
- **anim_en** -- LV_ANIM_ON: set with animation; LV_ANOM_OFF set immediately

void **lv_roller_set_visible_row_count**(*lv_obj_t* *obj, uint8_t row_cnt)
Set the height to show the given number of rows (options)

Parameters

- **obj** -- pointer to a roller object
- **row_cnt** -- number of desired visible rows

uint16_t **lv_roller_get_selected**(const *lv_obj_t* *obj)
Get the index of the selected option

Parameters **obj** -- pointer to a roller object

Returns index of the selected option (0 ... number of option - 1);

void **lv_roller_get_selected_str**(const *lv_obj_t* *obj, char *buf, uint32_t buf_size)
Get the current selected option as a string.

Parameters

- **obj** -- pointer to ddlist object
- **buf** -- pointer to an array to store the string
- **buf_size** -- size of `buf` in bytes. 0: to ignore it.

const char ***lv_roller_get_options**(const *lv_obj_t* *obj)
Get the options of a roller

Parameters **obj** -- pointer to roller object

Returns

the options separated by '

'-s (E.g. "Option1\nOption2\nOption3")

uint16_t **lv_roller_get_option_cnt**(const *lv_obj_t* *obj)
Get the total number of options

Parameters **obj** -- pointer to a roller object

Returns the total number of options

Variables

const lv_obj_class_t **lv_roller_class**

struct **lv_roller_t**

Public Members

lv_obj_t **obj**

uint16_t **option_cnt**
Number of options

uint16_t **sel_opt_id**
Index of the current option

uint16_t **sel_opt_id_ori**
Store the original index on focus

lv_roller_mode_t **mode**

uint32_t **moved**

5.2.12 Slider (lv_slider)

Overview

The Slider object looks like a *Bar* supplemented with a knob. The knob can be dragged to set a value. The Slider also can be vertical or horizontal.

Parts and Styles

- `LV_PART_MAIN` The background of the slider and it uses all the typical background style properties. `padding` makes the indicator smaller in the respective direction.
- `LV_PART_INDICATOR` The indicator the show 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. It also uses all the typical background properties to describe the knob(s). By default the knob is square (with a optional 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) the `lv_slider_set_range(slider, min , max)` can be used.

Modes

The slider can be one the following modes:

- `LV_SLIDER_MODE_NORMAL` A normal slider as described above
- `LV_SLIDER_SYMMETRICAL` Draw the indicator form the zero value to current value. Requires negaitve 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 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 only when it is released. Use `lv_slider_is_dragged` to decide whether is slider is being dragged or just released.

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

C

Simple Slider

code

```
#include "../../lv_examples.h"
#if LV_USE_SLIDER && LV_BUILD_EXAMPLES

static void slider_event_cb(lv_event_t * e);
static lv_obj_t * slider_label;

/**
 * A default slider with a label displaying the current value
 */
void lv_example_slider_1(void)
{
    /*Create a slider in the center of the display*/
    lv_obj_t * slider = lv_slider_create(lv_scr_act());
    lv_obj_center(slider);
    lv_obj_add_event_cb(slider, slider_event_cb, LV_EVENT_VALUE_CHANGED, NULL);

    /*Create a label below the slider*/
    slider_label = lv_label_create(lv_scr_act());
    lv_label_set_text(slider_label, "0%");

    lv_obj_align_to(slider_label, slider, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
}

static void slider_event_cb(lv_event_t * e)
{
    lv_obj_t * slider = lv_event_get_target(e);
    char buf[8];
    lv_snprintf(buf, sizeof(buf), "%d%%", lv_slider_get_value(slider));
    lv_label_set_text(slider_label, buf);
    lv_obj_align_to(slider_label, slider, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
}
```

(continues on next page)

(continued from previous page)

```
#endif
```

Slider with custom style

code

```
#include "../../lv_examples.h"
#if LV_USE_SLIDER && LV_BUILD_EXAMPLES



/**
 * Show how to style a slider.
 */
void lv_example_slider_2(void)
{
    /*Create a transition*/
    static const lv_style_prop_t props[] = {LV_STYLE_BG_COLOR, 0};
    static lv_style_transition_dsc_t transition_dsc;
    lv_style_transition_dsc_init(&transition_dsc, props, lv_anim_path_linear, 300, 0,␣
↪NULL);

    static lv_style_t style_main;
    static lv_style_t style_indicator;
    static lv_style_t style_knob;
    static lv_style_t style_pressed_color;
    lv_style_init(&style_main);
    lv_style_set_bg_opa(&style_main, LV_OPA_COVER);
    lv_style_set_bg_color(&style_main, lv_color_hex3(0xbbb));
    lv_style_set_radius(&style_main, LV_RADIUS_CIRCLE);
    lv_style_set_pad_ver(&style_main, -2); /*Makes the indicator larger*/

    lv_style_init(&style_indicator);
    lv_style_set_bg_opa(&style_indicator, LV_OPA_COVER);
    lv_style_set_bg_color(&style_indicator, lv_palette_main(LV_PALETTE_CYAN));
    lv_style_set_radius(&style_indicator, LV_RADIUS_CIRCLE);
    lv_style_set_transition(&style_indicator, &transition_dsc);

    lv_style_init(&style_knob);
    lv_style_set_bg_opa(&style_knob, LV_OPA_COVER);
    lv_style_set_bg_color(&style_knob, lv_palette_main(LV_PALETTE_CYAN));
    lv_style_set_border_color(&style_knob, lv_palette_darken(LV_PALETTE_CYAN, 3));
    lv_style_set_border_width(&style_knob, 2);
    lv_style_set_radius(&style_knob, LV_RADIUS_CIRCLE);
    lv_style_set_pad_all(&style_knob, 6); /*Makes the knob larger*/
    lv_style_set_transition(&style_knob, &transition_dsc);

    lv_style_init(&style_pressed_color);
    lv_style_set_bg_color(&style_pressed_color, lv_palette_darken(LV_PALETTE_CYAN,␣
↪2));

    /*Create a slider and add the style*/
    lv_obj_t * slider = lv_slider_create(lv_scr_act());
    lv_obj_remove_style_all(slider);        /*Remove the styles coming from the␣
↪theme*/
```

(continues on next page)

(continued from previous page)

```
    lv_obj_add_style(slider, &style_main, LV_PART_MAIN);
    lv_obj_add_style(slider, &style_indicator, LV_PART_INDICATOR);
    lv_obj_add_style(slider, &style_pressed_color, LV_PART_INDICATOR | LV_STATE_
↪PRESSED);
    lv_obj_add_style(slider, &style_knob, LV_PART_KNOB);
    lv_obj_add_style(slider, &style_pressed_color, LV_PART_KNOB | LV_STATE_PRESSED);

    lv_obj_center(slider);
}

#endif
```

Slider with extended drawer

code

```
#include "../../lv_examples.h"
#if LV_USE_SLIDER && LV_BUILD_EXAMPLES

static void slider_event_cb(lv_event_t * e);

/**
 * Show the current value when the slider is pressed by extending the drawer
 *
 */
void lv_example_slider_3(void)
{
    /*Create a slider in the center of the display*/
    lv_obj_t * slider;
    slider = lv_slider_create(lv_scr_act());
    lv_obj_center(slider);

    lv_slider_set_mode(slider, LV_SLIDER_MODE_RANGE);
    lv_slider_set_value(slider, 70, LV_ANIM_OFF);
    lv_slider_set_left_value(slider, 20, LV_ANIM_OFF);

    lv_obj_add_event_cb(slider, slider_event_cb, LV_EVENT_ALL, NULL);
    lv_obj_refresh_ext_draw_size(slider);
}

static void slider_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target(e);

    /*Provide some extra space for the value*/
    if(code == LV_EVENT_REFR_EXT_DRAW_SIZE) {
        lv_coord_t * size = lv_event_get_param(e);
        *size = LV_MAX(*size, 50);
    }
    else if(code == LV_EVENT_DRAW_PART_END) {
        lv_obj_draw_part_dsc_t * dsc = lv_event_get_param(e);
        if(dsc->part == LV_PART_INDICATOR) {
            char buf[16];
```

(continues on next page)

(continued from previous page)

```
            lv_snprintf(buf, sizeof(buf), "%d - %d", lv_slider_get_left_value(obj),␣
↪lv_slider_get_value(obj));

            lv_point_t label_size;
            lv_txt_get_size(&label_size, buf, LV_FONT_DEFAULT, 0, 0, LV_COORD_MAX, 0);
            lv_area_t label_area;
            label_area.x1 = dsc->draw_area->x1 + lv_area_get_width(dsc->draw_area) /␣
↪2 - label_size.x / 2;
            label_area.x2 = label_area.x1 + label_size.x;
            label_area.y2 = dsc->draw_area->y1 - 10;
            label_area.y1 = label_area.y2 - label_size.y;

            lv_draw_label_dsc_t label_draw_dsc;
            lv_draw_label_dsc_init(&label_draw_dsc);

            lv_draw_label(&label_area, dsc->clip_area, &label_draw_dsc, buf, NULL);
        }
    }
}

#endif
```

MicroPython

No examples yet.

API

Typedefs

typedef uint8_t **lv_slider_mode_t**

Enums

enum **[anonymous]**

Values:

enumerator **LV_SLIDER_MODE_NORMAL**

enumerator **LV_SLIDER_MODE_SYMMETRICAL**

enumerator **LV_SLIDER_MODE_RANGE**

Functions

lv_obj_t ***lv_slider_create**(*lv_obj_t* *parent)
Create a slider objects

Parameters **parent** -- pointer to an object, it will be the parent of the new slider.

Returns pointer to the created slider

static inline void **lv_slider_set_value**(*lv_obj_t* *obj, int32_t value, *lv_anim_enable_t* anim)
Set a new value on the slider

Parameters

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

static inline void **lv_slider_set_left_value**(*lv_obj_t* *obj, int32_t value, *lv_anim_enable_t* anim)
Set a new value for the left knob of a slider

Parameters

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

static inline void **lv_slider_set_range**(*lv_obj_t* *obj, int32_t min, int32_t max)
Set minimum and the maximum values of a bar

Parameters

- **obj** -- pointer to the slider object
- **min** -- minimum value
- **max** -- maximum value

static inline void **lv_slider_set_mode**(*lv_obj_t* *obj, *lv_slider_mode_t* mode)
Set the mode of slider.

Parameters

- **obj** -- pointer to a slider object
- **mode** -- the mode of the slider. See ::lv_slider_mode_t

static inline int32_t **lv_slider_get_value**(const *lv_obj_t* *obj)
Get the value of the main knob of a slider

Parameters **obj** -- pointer to a slider object

Returns the value of the main knob of the slider

static inline int32_t **lv_slider_get_left_value**(const *lv_obj_t* *obj)
Get the value of the left knob of a slider

Parameters **obj** -- pointer to a slider object

Returns the value of the left knob of the slider

static inline int32_t **lv_slider_get_min_value**(const *lv_obj_t* *obj)
Get the minimum value of a slider

Parameters **obj** -- pointer to a slider object

Returns the minimum value of the slider

static inline int32_t **lv_slider_get_max_value**(const *lv_obj_t* *obj)
Get the maximum value of a slider

Parameters **obj** -- pointer to a slider object

Returns the maximum value of the slider

bool **lv_slider_is_dragged**(const *lv_obj_t* *obj)
Give the slider is being dragged or not

Parameters **obj** -- pointer to a slider object

Returns true: drag in progress false: not dragged

static inline *lv_slider_mode_t* **lv_slider_get_mode**(*lv_obj_t* *slider)
Get the mode of the slider.

Parameters **obj** -- pointer to a bar object

Returns see ::lv_slider_mode_t

Variables

const lv_obj_class_t **lv_slider_class**

struct **lv_slider_t**

Public Members

lv_bar_t **bar**

lv_area_t **left_knob_area**

lv_area_t **right_knob_area**

int32_t ***value_to_set**

uint8_t **dragging**

uint8_t **left_knob_focus**

5.2.13 Switch (lv_switch)

Overview

The Switch can be used to turn on/off something. It looks like a little slider.

Parts and Styles

- `LV_PART_MAIN` The background of the switch and it uses all the typical background style properties. `padding` makes the indicator smaller in the respective direction.
- `LV_PART_INDICATOR` The indicator the show 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 teh indicator. It also uses all the typical background properties to describe the knob(s). By default the knob is square (with a optional 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_CHACKED`. To get the current satte of the switch use `lv_obj_has_state(switch, LV_STATE_CHECHKED)`. To manually turn the switch on/off call `lvobj_add/clear_state(switch, LV_STATE_CHECKED)`.

Events

- `LV_EVENT_VALUE_CHANGED` Sent when the switch changes state.

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

C

Simple Switch

code

```
Error encountered while trying to open /home/runner/work/100ask_lvgl_docs_8.x/100ask_
↪lvgl_docs_8.x/examples/widgets/lv_example_switch/lv_example_switch_1.c
```

MicroPython

No examples yet.

API

Functions

lv_obj_t ***lv_switch_create**(*lv_obj_t* *parent)

Create a switch objects

Parameters **parent** -- pointer to an object, it will be the parent of the new switch

Returns pointer to the created switch

Variables

const lv_obj_class_t **lv_switch_class**

struct **lv_switch_t**

Public Members

lv_obj_t **obj**

5.2.14 Table (lv_table)

Overview

Tables, as usual, are built from rows, columns, and cells containing texts.

The Table object is very light weighted because only the texts are stored. No real objects are created for cells but they are just drawn on the fly.

Parts and Styles

- `LV_PART_MAIN` The background of the table and uses all the typical background style properties.
- `LV_PART_ITEMS` The cells of the table and they also use all the typical background style properties and the text properties.

Usage

Set cell value

The cells can store only texts so numbers needs to be converted to text before displaying them in a table.

`lv_table_set_cell_value(table, row, col, "Content")`. The text is saved by the table so it can be even a local variable.

Line break can be used in the text like `"Value\n60.3"`.

The new rows and column 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_set_cell_merge_right(table, col, row, true)`. To merge more adjacent cells apply this function for each cell.

Scroll

If the label's width or height is set to `LV_SIZE_CONTENT` that size will be se 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 smaller number than the "intrinsic" size then the table becomes scrollable.

Events

- `LV_EVENT_DRAW_PART_BEGIN` and `LV_EVENT_DRAW_PART_END` are sent for both main and items parts to allow hooking the drawing. The for more detail on the main part see the Base object's documentation. For the items (sells) the following fields are used: `clip_area`, `draw_area`, `part`, `rect_dsc`, `label_dsc` `id` (current row × col count + current column).

Learn more about *Events*.

Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

Example

C

Simple table

code

```
#include "../../lv_examples.h"
#if LV_USE_TABLE && LV_BUILD_EXAMPLES

static void draw_part_event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target(e);
    lv_obj_draw_part_dsc_t * dsc = lv_event_get_param(e);
    /*If the cells are drawn...*/
    if(dsc->part == LV_PART_ITEMS) {
        uint32_t row = dsc->id /  lv_table_get_col_cnt(obj);
        uint32_t col = dsc->id - row * lv_table_get_col_cnt(obj);

        /*Make the texts in the first cell center aligned*/
        if(row == 0) {
            dsc->label_dsc->align = LV_TEXT_ALIGN_CENTER;
            dsc->rect_dsc->bg_color = lv_color_mix(lv_palette_main(LV_PALETTE_BLUE),␣
↪dsc->rect_dsc->bg_color, LV_OPA_20);
            dsc->rect_dsc->bg_opa = LV_OPA_COVER;
        }
        /*In the first column align the texts to the right*/
        else if(col == 0) {
            dsc->label_dsc->flag = LV_TEXT_ALIGN_RIGHT;
        }

        /*MAke every 2nd row grayish*/
        if((row != 0 && row % 2) == 0) {
            dsc->rect_dsc->bg_color = lv_color_mix(lv_palette_main(LV_PALETTE_GREY),␣
↪dsc->rect_dsc->bg_color, LV_OPA_10);
            dsc->rect_dsc->bg_opa = LV_OPA_COVER;
        }
    }
}


void lv_example_table_1(void)
{
    lv_obj_t * table = lv_table_create(lv_scr_act());

    /*Fill the first column*/
    lv_table_set_cell_value(table, 0, 0, "Name");
    lv_table_set_cell_value(table, 1, 0, "Apple");
    lv_table_set_cell_value(table, 2, 0, "Banana");
```

(continues on next page)

(continued from previous page)

```
    lv_table_set_cell_value(table, 3, 0, "Lemon");
    lv_table_set_cell_value(table, 4, 0, "Grape");
    lv_table_set_cell_value(table, 5, 0, "Melon");
    lv_table_set_cell_value(table, 6, 0, "Peach");
    lv_table_set_cell_value(table, 7, 0, "Nuts");

    /*Fill the second column*/
    lv_table_set_cell_value(table, 0, 1, "Price");
    lv_table_set_cell_value(table, 1, 1, "$7");
    lv_table_set_cell_value(table, 2, 1, "$4");
    lv_table_set_cell_value(table, 3, 1, "$6");
    lv_table_set_cell_value(table, 4, 1, "$2");
    lv_table_set_cell_value(table, 5, 1, "$5");
    lv_table_set_cell_value(table, 6, 1, "$1");
    lv_table_set_cell_value(table, 7, 1, "$9");

    /*Set a smaller height to the table. It'll make it scrollable*/
    lv_obj_set_height(table, 200);
    lv_obj_center(table);

    /*Add an event callback to to apply some custom drawing*/
    lv_obj_add_event_cb(table, draw_part_event_cb, LV_EVENT_DRAW_PART_BEGIN, NULL);
}

#endif
```

Lightweighted list from table

code

```
#include "../../lv_examples.h"
#if LV_USE_TABLE && LV_BUILD_EXAMPLES

#define ITEM_CNT 200

static void draw_event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target(e);
    lv_obj_draw_part_dsc_t * dsc = lv_event_get_draw_part_dsc(e);
    /*If the cells are drawn...*/
    if(dsc->part == LV_PART_ITEMS) {
        bool chk = lv_table_has_cell_ctrl(obj, dsc->id, 0, LV_TABLE_CELL_CTRL_CUSTOM_
↪1);

        lv_draw_rect_dsc_t rect_dsc;
        lv_draw_rect_dsc_init(&rect_dsc);
        rect_dsc.bg_color = chk ? lv_theme_get_color_primary(obj) : lv_palette_
↪lighten(LV_PALETTE_GREY, 2);
        rect_dsc.radius = LV_RADIUS_CIRCLE;

        lv_area_t sw_area;
        sw_area.x1 = dsc->draw_area->x2 - 50;
        sw_area.x2 = sw_area.x1 + 40;
        sw_area.y1 =  dsc->draw_area->y1 + lv_area_get_height(dsc->draw_area) / 2 -␣
↪10;
```

(continues on next page)

(continued from previous page)

```
        sw_area.y2 = sw_area.y1 + 20;
        lv_draw_rect(&sw_area, dsc->clip_area, &rect_dsc);

        rect_dsc.bg_color = lv_color_white();
        if(chk) {
            sw_area.x2 -= 2;
            sw_area.x1 = sw_area.x2 - 16;
        } else {
            sw_area.x1 += 2;
            sw_area.x2 = sw_area.x1 + 16;
        }
        sw_area.y1 += 2;
        sw_area.y2 -= 2;
        lv_draw_rect(&sw_area, dsc->clip_area, &rect_dsc);
    }
}

static void change_event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target(e);
    uint16_t col;
    uint16_t row;
    lv_table_get_selected_cell(obj, &row, &col);
    bool chk = lv_table_has_cell_ctrl(obj, row, 0, LV_TABLE_CELL_CTRL_CUSTOM_1);
    if(chk) lv_table_clear_cell_ctrl(obj, row, 0, LV_TABLE_CELL_CTRL_CUSTOM_1);
    else lv_table_add_cell_ctrl(obj, row, 0, LV_TABLE_CELL_CTRL_CUSTOM_1);
}


/**
 * A very light-weighted list created from table
 */
void lv_example_table_2(void)
{
    /*Measure memory usage*/
    lv_mem_monitor_t mon1;
    lv_mem_monitor(&mon1);

    uint32_t t = lv_tick_get();

    lv_obj_t * table = lv_table_create(lv_scr_act());

    /*Set a smaller height to the table. It'll make it scrollable*/
    lv_obj_set_size(table, 150, 200);

    lv_table_set_col_width(table, 0, 150);
    lv_table_set_row_cnt(table, ITEM_CNT); /*Not required but avoids a lot of memory␣
↪reallocation lv_table_set_set_value*/
    lv_table_set_col_cnt(table, 1);

    /*Don't make the cell pressed, we will draw something different in the event*/
    lv_obj_remove_style(table, NULL, LV_PART_ITEMS | LV_STATE_PRESSED);

    uint32_t i;
    for(i = 0; i < ITEM_CNT; i++) {
        lv_table_set_cell_value_fmt(table, i, 0, "Item %d", i + 1);
    }
```

(continues on next page)

(continued from previous page)

```
    lv_obj_align(table, LV_ALIGN_CENTER, 0, -20);

    /*Add an event callback to to apply some custom drawing*/
    lv_obj_add_event_cb(table, draw_event_cb, LV_EVENT_DRAW_PART_END, NULL);
    lv_obj_add_event_cb(table, change_event_cb, LV_EVENT_VALUE_CHANGED, NULL);

    lv_mem_monitor_t mon2;
    lv_mem_monitor(&mon2);

    uint32_t mem_used = mon1.free_size - mon2.free_size;

    uint32_t elaps = lv_tick_elaps(t);

    lv_obj_t * label = lv_label_create(lv_scr_act());
    lv_label_set_text_fmt(label, "%d items were created in %d ms\n"
                                 "using %d bytes of memory",
                                 ITEM_CNT, elaps, mem_used);

    lv_obj_align(label, LV_ALIGN_BOTTOM_MID, 0, -10);

}

#endif
```

MicroPython

No examples yet.

MicroPython

No examples yet.

API

Typedefs

typedef uint8_t **lv_table_cell_ctrl_t**

Enums

enum **[anonymous]**

Values:

enumerator **LV_TABLE_CELL_CTRL_MERGE_RIGHT**

enumerator **LV_TABLE_CELL_CTRL_TEXT_CROP**

enumerator **LV_TABLE_CELL_CTRL_CUSTOM_1**

enumerator **LV_TABLE_CELL_CTRL_CUSTOM_2**

enumerator **LV_TABLE_CELL_CTRL_CUSTOM_3**

enumerator **LV_TABLE_CELL_CTRL_CUSTOM_4**

Functions

LV_EXPORT_CONST_INT(LV_TABLE_CELL_NONE)

lv_obj_t ***lv_table_create**(*lv_obj_t* *parent)

Create a table object

Parameters **parent** -- pointer to an object, it will be the parent of the new table

Returns pointer to the created table

void **lv_table_set_cell_value**(*lv_obj_t* *obj, uint16_t row, uint16_t col, const char *txt)

Set the value of a cell.

Note: New roes/columns are added automatically if required

Parameters

- **obj** -- pointer to a Table object
- **row** -- id of the row [0 .. row_cnt -1]
- **col** -- id of the column [0 .. col_cnt -1]
- **txt** -- text to display in the cell. It will be copied and saved so this variable is not required after this function call.

void **lv_table_set_cell_value_fmt**(*lv_obj_t* *obj, uint16_t row, uint16_t col, const char *fmt, ...)

Set the value of a cell. Memory will be allocated to store the text by the table.

Note: New roes/columns are added automatically if required

Parameters

- **obj** -- pointer to a Table object
- **row** -- id of the row [0 .. row_cnt -1]
- **col** -- id of the column [0 .. col_cnt -1]
- **fmt** -- `printf`-like format

void **lv_table_set_row_cnt**(*lv_obj_t* *obj, uint16_t row_cnt)

Set the number of rows

Parameters

- **obj** -- table pointer to a Table object
- **row_cnt** -- number of rows

void **lv_table_set_col_cnt**(*lv_obj_t* *obj, uint16_t col_cnt)
Set the number of columns

Parameters

- **obj** -- table pointer to a Table object
- **col_cnt** -- number of columns.

void **lv_table_set_col_width**(*lv_obj_t* *obj, uint16_t col_id, lv_coord_t w)
Set the width of a column

Parameters

- **obj** -- table pointer to a Table object
- **col_id** -- id of the column [0 .. LV_TABLE_COL_MAX -1]
- **w** -- width of the column

void **lv_table_add_cell_ctrl**(*lv_obj_t* *obj, uint16_t row, uint16_t col, *lv_table_cell_ctrl_t* ctrl)
Add control bits to the cell.

Parameters

- **obj** -- pointer to a Table object
- **row** -- id of the row [0 .. row_cnt -1]
- **col** -- id of the column [0 .. col_cnt -1]
- **ctrl** -- OR-ed values from ::lv_table_cell_ctrl_t

void **lv_table_clear_cell_ctrl**(*lv_obj_t* *obj, uint16_t row, uint16_t col, *lv_table_cell_ctrl_t* ctrl)
Clear control bits of the cell.

Parameters

- **obj** -- pointer to a Table object
- **row** -- id of the row [0 .. row_cnt -1]
- **col** -- id of the column [0 .. col_cnt -1]
- **ctrl** -- OR-ed values from ::lv_table_cell_ctrl_t

const char ***lv_table_get_cell_value**(*lv_obj_t* *obj, uint16_t row, uint16_t col)
Get the value of a cell.

Parameters

- **obj** -- pointer to a Table object
- **row** -- id of the row [0 .. row_cnt -1]
- **col** -- id of the column [0 .. col_cnt -1]

Returns text in the cell

uint16_t **lv_table_get_row_cnt**(*lv_obj_t* *obj)
Get the number of rows.

Parameters **obj** -- table pointer to a Table object

Returns number of rows.

uint16_t **lv_table_get_col_cnt**(*lv_obj_t* *obj)
Get the number of columns.

Parameters **obj** -- table pointer to a Table object

Returns number of columns.

lv_coord_t **lv_table_get_col_width**(*lv_obj_t* *obj, uint16_t col)

Get the width of a column

Parameters

- **obj** -- table pointer to a Table object
- **col** -- id of the column [0 .. LV_TABLE_COL_MAX -1]

Returns width of the column

bool **lv_table_has_cell_ctrl**(*lv_obj_t* *obj, uint16_t row, uint16_t col, *lv_table_cell_ctrl_t* ctrl)

Get whether a cell has the control bits

Parameters

- **obj** -- pointer to a Table object
- **row** -- id of the row [0 .. row_cnt -1]
- **col** -- id of the column [0 .. col_cnt -1]
- **ctrl** -- OR-ed values from ::lv_table_cell_ctrl_t

Returns true: all control bits are set; false: not all control bits are set

void **lv_table_get_selected_cell**(*lv_obj_t* *obj, uint16_t *row, uint16_t *col)

Get the selected cell (pressed and or focused)

Parameters

- **obj** -- pointer to a table object
- **row** -- pointer to variable to store the selected row (LV_TABLE_CELL_NONE: if no cell selected)
- **col** -- pointer to variable to store the selected column (LV_TABLE_CELL_NONE: if no cell selected)

Variables

const lv_obj_class_t **lv_table_class**

struct **lv_table_t**

Public Members

lv_obj_t **obj**

uint16_t **col_cnt**

uint16_t **row_cnt**

char ****cell_data**

lv_coord_t ***row_h**

lv_coord_t ***col_w**

uint16_t **col_act**

uint16_t **row_act**

5.2.15 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 and it uses all the typical backgrond 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` Tells the style of the *selected text*. Only `text_color` and `bg_color` style properties can be used.
- `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 let the cursor transparent and set a left border. The `anim_time` style property sets the cursors blink time.
- `LV_PART_TEXTAREA_PLACEHOLDER` It's a part related only to the text area and 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, hiwever it can be good idea to style to be visible only in `LV_STATE_FOCUSED` state.

One line mode

The Text area can be configures to be one lined with `lv_textarea_set_one_line(textarea, true)`. In this mode the height is set automatically to show only one line, line break character are ignored, and word wrap is disabled.

Password mode

The text area supports password mode which can be enabled with `lv_textarea_set_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)` gives the real text, 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_textarae_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) its 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. It will save some information about the label to speed up its drawing. Using `LV_LABEL_LONG_TXT_HINT` the scrolling and drawing will as fast as with "normal" short texts.

Select text

A part of text can be selected if enabled with `lv_textarea_set_text_selection(textarea, true)`. It works like when you select a text on your PC with your mouse.

Events

- `LV_EVENT_INSERT` Sent when before a character or text is inserted. The event paramter is the text planned to be inserted. `lv_textarea_set_insert_replace(textarea, "New text")` replaces the text to insert. The new text can not 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_APPLY` Sent when `LV_KEY_ENTER` is pressed (or(sent) to a one line text area.

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

C

Simple Text area

code

```
#include "../../lv_examples.h"
#if LV_USE_TEXTAREA && LV_BUILD_EXAMPLES

static void textarea_event_handler(lv_event_t * e)
{
    lv_obj_t * ta = lv_event_get_target(e);
    LV_LOG_USER("Enter was pressed. The current text is: %s", lv_textarea_get_
↪text(ta));
}

static void btnm_event_handler(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target(e);
    lv_obj_t * ta = lv_event_get_user_data(e);
    const char * txt = lv_btnmatrix_get_btn_text(obj, lv_btnmatrix_get_selected_
↪btn(obj));

    if(strcmp(txt, LV_SYMBOL_BACKSPACE) == 0) lv_textarea_del_char(ta);
    else if(strcmp(txt, LV_SYMBOL_NEW_LINE) == 0) lv_textarea_add_char(ta, '\n');
    else lv_textarea_add_text(ta, txt);

}

void lv_example_textarea_1(void)
{
    lv_obj_t * ta = lv_textarea_create(lv_scr_act());
    lv_textarea_set_one_line(ta, true);
    lv_obj_align(ta, LV_ALIGN_TOP_MID, 0, 10);
    lv_obj_add_event_cb(ta, textarea_event_handler, LV_EVENT_READY, ta);
    lv_obj_add_state(ta, LV_STATE_FOCUSED); /*To be sure the cursor is visible*/

    static const char * btnm_map[] = {"1", "2", "3", "\n",
                                      "4", "5", "6", "\n",
                                      "7", "8", "9", "\n",
                                      LV_SYMBOL_BACKSPACE, "0", LV_SYMBOL_NEW_LINE, ""};

    lv_obj_t * btnm = lv_btnmatrix_create(lv_scr_act());
    lv_obj_set_size(btnm, 200, 150);
    lv_obj_align(btnm, LV_ALIGN_BOTTOM_MID, 0, -10);
    lv_obj_add_event_cb(btnm, btnm_event_handler, LV_EVENT_VALUE_CHANGED, ta);
    lv_obj_clear_flag(btnm, LV_OBJ_FLAG_CLICK_FOCUSABLE); /*To keep the text area␣
↪focused on button clicks*/
    lv_btnmatrix_set_map(btnm, btnm_map);
}

#endif
```

Text area with password field

code

```
#include "../../lv_examples.h"
#if LV_USE_TEXTAREA && LV_USE_KEYBOARD && LV_BUILD_EXAMPLES

static void ta_event_cb(lv_event_t * e);

static lv_obj_t * kb;

void lv_example_textarea_2(void)
{
    /*Create the password box*/
    lv_obj_t * pwd_ta = lv_textarea_create(lv_scr_act());
    lv_textarea_set_text(pwd_ta, "");
    lv_textarea_set_password_mode(pwd_ta, true);
    lv_textarea_set_one_line(pwd_ta, true);
    lv_obj_set_width(pwd_ta, lv_pct(40));
    lv_obj_set_pos(pwd_ta, 5, 20);
    lv_obj_add_event_cb(pwd_ta, ta_event_cb, LV_EVENT_ALL, NULL);

    /*Create a label and position it above the text box*/
    lv_obj_t * pwd_label = lv_label_create(lv_scr_act());
    lv_label_set_text(pwd_label, "Password:");
    lv_obj_align_to(pwd_label, pwd_ta, LV_ALIGN_OUT_TOP_LEFT, 0, 0);

    /*Create the one-line mode text area*/
    lv_obj_t * text_ta = lv_textarea_create(lv_scr_act());
    lv_textarea_set_one_line(text_ta, true);
    lv_textarea_set_password_mode(text_ta, false);
    lv_obj_set_width(text_ta, lv_pct(40));
    lv_obj_add_event_cb(text_ta, ta_event_cb, LV_EVENT_ALL, NULL);
    lv_obj_align(text_ta, LV_ALIGN_TOP_RIGHT, -5, 20);


    /*Create a label and position it above the text box*/
    lv_obj_t * oneline_label = lv_label_create(lv_scr_act());
    lv_label_set_text(oneline_label, "Text:");
    lv_obj_align_to(oneline_label, text_ta, LV_ALIGN_OUT_TOP_LEFT, 0, 0);

    /*Create a keyboard*/
    kb = lv_keyboard_create(lv_scr_act());
    lv_obj_set_size(kb,  LV_HOR_RES, LV_VER_RES / 2);

    lv_keyboard_set_textarea(kb, pwd_ta); /*Focus it on one of the text areas to␣
↪start*/
}

static void ta_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * ta = lv_event_get_target(e);
    if(code == LV_EVENT_CLICKED || code == LV_EVENT_FOCUSED) {
        /*Focus on the clicked text area*/
        if(kb != NULL) lv_keyboard_set_textarea(kb, ta);
    }
```

(continues on next page)

(continued from previous page)

```
    else if(code == LV_EVENT_READY) {
        const char * str = lv_event_get_param(e);
        if(str[0] == '\n') {
            LV_LOG_USER("Ready\n");
        }
    }
}

#endif
```

Text auto-formatting

code

```
#include "../../lv_examples.h"
#if LV_USE_TEXTAREA && LV_USE_KEYBOARD && LV_BUILD_EXAMPLES

static void ta_event_cb(lv_event_t * e);

static lv_obj_t * kb;

/**
 * Automatically format text like a clock. E.g. "12:34"
 * Add the ':' automatically.
 */
void lv_example_textarea_3(void)
{
    /*Create the text area*/
    lv_obj_t * ta = lv_textarea_create(lv_scr_act());
    lv_obj_add_event_cb(ta, ta_event_cb, LV_EVENT_VALUE_CHANGED, NULL);
    lv_textarea_set_accepted_chars(ta, "0123456789:");
    lv_textarea_set_max_length(ta, 5);
    lv_textarea_set_one_line(ta, true);
    lv_textarea_set_text(ta, "");

    /*Create a keyboard*/
    kb = lv_keyboard_create(lv_scr_act());
    lv_obj_set_size(kb,  LV_HOR_RES, LV_VER_RES / 2);
    lv_keyboard_set_mode(kb, LV_KEYBOARD_MODE_NUMBER);
    lv_keyboard_set_textarea(kb, ta);
}

static void ta_event_cb(lv_event_t * e)
{
    lv_obj_t * ta = lv_event_get_target(e);
    const char * txt = lv_textarea_get_text(ta);
    if(txt[0] >= '0' && txt[0] <= '9' &&
        txt[1] >= '0' && txt[1] <= '9' &&
        txt[2] != ':')
    {
        lv_textarea_set_cursor_pos(ta, 2);
        lv_textarea_add_char(ta, ':');
    }
}
```

(continues on next page)

(continued from previous page)

```
#endif
```

MicroPython

No examples yet.

API

Enums

enum **[anonymous]**
Values:

enumerator **LV_PART_TEXTAREA_PLACEHOLDER**

Functions

LV_EXPORT_CONST_INT(LV_TEXTAREA_CURSOR_LAST)

lv_obj_t ***lv_textarea_create**(*lv_obj_t* *parent)
Create a text area objects

Parameters **parent** -- pointer to an object, it will be the parent of the new text area

Returns pointer to the created text area

void **lv_textarea_add_char**(*lv_obj_t* *obj, uint32_t c)
Insert a character to the current cursor position. To add a wide char, e.g. 'Á' use _lv_txt_encoded_conv_wc('Á)`

Parameters

- **obj** -- pointer to a text area object
- **c** -- a character (e.g. 'a')

void **lv_textarea_add_text**(*lv_obj_t* *obj, const char *txt)
Insert a text to the current cursor position

Parameters

- **obj** -- pointer to a text area object
- **txt** -- a '\0' terminated string to insert

void **lv_textarea_del_char**(*lv_obj_t* *obj)
Delete a the left character from the current cursor position

Parameters **obj** -- pointer to a text area object

void **lv_textarea_del_char_forward**(*lv_obj_t* *obj)
Delete the right character from the current cursor position

Parameters **obj** -- pointer to a text area object

void **lv_textarea_set_text**(*lv_obj_t* *obj, const char *txt)
Set the text of a text area

Parameters

- **obj** -- pointer to a text area object
- **txt** -- pointer to the text

void **lv_textarea_set_placeholder_text**(*lv_obj_t* *obj, const char *txt)
Set the placeholder text of a text area

Parameters

- **obj** -- pointer to a text area object
- **txt** -- pointer to the text

void **lv_textarea_set_cursor_pos**(*lv_obj_t* *obj, int32_t pos)
Set the cursor position

Parameters

- **obj** -- pointer to a text area object
- **pos** -- the new cursor position in character index < 0 : index from the end of the text LV_TEXTAREA_CURSOR_LAST: go after the last character

void **lv_textarea_set_cursor_click_pos**(*lv_obj_t* *obj, bool en)
Enable/Disable the positioning of the cursor by clicking the text on the text area.

Parameters

- **obj** -- pointer to a text area object
- **en** -- true: enable click positions; false: disable

void **lv_textarea_set_password_mode**(*lv_obj_t* *obj, bool en)
Enable/Disable password mode

Parameters

- **obj** -- pointer to a text area object
- **en** -- true: enable, false: disable

void **lv_textarea_set_one_line**(*lv_obj_t* *obj, bool en)
Configure the text area to one line or back to normal

Parameters

- **obj** -- pointer to a text area object
- **en** -- true: one line, false: normal

void **lv_textarea_set_accepted_chars**(*lv_obj_t* *obj, const char *list)
Set a list of characters. Only these characters will be accepted by the text area

Parameters

- **obj** -- pointer to a text area object
- **list** -- list of characters. Only the pointer is saved. E.g. "+-.,0123456789"

void **lv_textarea_set_max_length**(*lv_obj_t* *obj, uint32_t num)
Set max length of a Text Area.

Parameters

- **obj** -- pointer to a text area object
- **num** -- the maximal number of characters can be added (`lv_textarea_set_text` ignores it)

void **lv_textarea_set_insert_replace**(*lv_obj_t* *obj, const char *txt)

In `LV_EVENT_INSERT` the text which planned to be inserted can be replaced by an other text. It can be used to add automatic formatting to the text area.

Parameters

- **obj** -- pointer to a text area object
- **txt** -- pointer to a new string to insert. If `""` no text will be added. The variable must be live after the `event_cb` exists. (Should be `global` or `static`)

void **lv_textarea_set_text_selection**(*lv_obj_t* *obj, bool en)

Enable/disable selection mode.

Parameters

- **obj** -- pointer to a text area object
- **en** -- true or false to enable/disable selection mode

void **lv_textarea_set_password_show_time**(*lv_obj_t* *obj, uint16_t time)

Set how long show the password before changing it to '*'

Parameters

- **obj** -- pointer to a text area object
- **time** -- show time in milliseconds. 0: hide immediately.

void **lv_textarea_set_align**(*lv_obj_t* *obj, lv_text_align_t align)

Set the label's alignment. It sets where the label is aligned (in one line mode it can be smaller than the text area) and how the lines of the area align in case of multiline text area

Parameters

- **obj** -- pointer to a text area object
- **align** -- the align mode from ::lv_text_align_t

const char ***lv_textarea_get_text**(const *lv_obj_t* *obj)

Get the text of a text area. In password mode it gives the real text (not '*'s).

Parameters **obj** -- pointer to a text area object

Returns pointer to the text

const char ***lv_textarea_get_placeholder_text**(*lv_obj_t* *obj)

Get the placeholder text of a text area

Parameters **obj** -- pointer to a text area object

Returns pointer to the text

lv_obj_t ***lv_textarea_get_label**(const *lv_obj_t* *obj)

Get the label of a text area

Parameters **obj** -- pointer to a text area object

Returns pointer to the label object

uint32_t **lv_textarea_get_cursor_pos**(const *lv_obj_t* *obj)

Get the current cursor position in character index

Parameters **obj** -- pointer to a text area object

Returns the cursor position

bool **lv_textarea_get_cursor_click_pos**(*lv_obj_t* *obj)
Get whether the cursor click positioning is enabled or not.

Parameters **obj** -- pointer to a text area object

Returns true: enable click positions; false: disable

bool **lv_textarea_get_password_mode**(const *lv_obj_t* *obj)
Get the password mode attribute

Parameters **obj** -- pointer to a text area object

Returns true: password mode is enabled, false: disabled

bool **lv_textarea_get_one_line**(const *lv_obj_t* *obj)
Get the one line configuration attribute

Parameters **obj** -- pointer to a text area object

Returns true: one line configuration is enabled, false: disabled

const char ***lv_textarea_get_accepted_chars**(*lv_obj_t* *obj)
Get a list of accepted characters.

Parameters **obj** -- pointer to a text area object

Returns list of accented characters.

uint32_t **lv_textarea_get_max_length**(*lv_obj_t* *obj)
Get max length of a Text Area.

Parameters **obj** -- pointer to a text area object

Returns the maximal number of characters to be add

bool **lv_textarea_text_is_selected**(const *lv_obj_t* *obj)
Find whether text is selected or not.

Parameters **obj** -- pointer to a text area object

Returns whether text is selected or not

bool **lv_textarea_get_text_selection**(*lv_obj_t* *obj)
Find whether selection mode is enabled.

Parameters **obj** -- pointer to a text area object

Returns true: selection mode is enabled, false: disabled

uint16_t **lv_textarea_get_password_show_time**(*lv_obj_t* *obj)
Set how long show the password before changing it to '*'

Parameters **obj** -- pointer to a text area object

Returns show time in milliseconds. 0: hide immediately.

void **lv_textarea_clear_selection**(*lv_obj_t* *obj)
Clear the selection on the text area.

Parameters **obj** -- pointer to a text area object

void **lv_textarea_cursor_right**(*lv_obj_t* *obj)
Move the cursor one character right

Parameters **obj** -- pointer to a text area object

void **lv_textarea_cursor_left**(*lv_obj_t* *obj)
Move the cursor one character left

Parameters **obj** -- pointer to a text area object

void **lv_textarea_cursor_down**(*lv_obj_t* *obj)
Move the cursor one line down

Parameters **obj** -- pointer to a text area object

void **lv_textarea_cursor_up**(*lv_obj_t* *obj)
Move the cursor one line up

Parameters **obj** -- pointer to a text area object

Variables

const lv_obj_class_t **lv_textarea_class**

struct **lv_textarea_t**

Public Members

lv_obj_t **obj**

lv_obj_t ***label**

char ***placeholder_txt**

char ***pwd_tmp**

const char ***accepted_chars**

uint32_t **max_length**

uint16_t **pwd_show_time**

lv_coord_t **valid_x**

uint32_t **pos**

lv_area_t **area**

uint32_t **txt_byte_pos**

uint8_t **show**

uint8_t **click_pos**

struct *lv_textarea_t*::[anonymous] **cursor**

uint32_t **sel_start**

uint32_t **sel_end**

uint8_t **text_sel_in_prog**

uint8_t **text_sel_en**

uint8_t **pwd_mode**

uint8_t **one_line**

5.3 Extra widgets

5.3.1 Calendar (lv_calendar)

Overview

The Calendar object is a classic calendar which can:

- can show the days of any month in a 7x7 matrix
- Show the name of the days
- highlight the current day
- highlight any user-defined dates

The Calendar is added to the deafult group (if it is set). Besides the Calendar is an editable object to allow selecting and clicing the dates with encoder navigation too.

To make the Calendar flexible, by default it doesn't show the curent year or month. Instead, there external "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`
- `LV_PART_ITEMS` Refers to the dates and day names. Button matrix control flags are set the to differentiate the buttons and a custom drawer event modifies the properties of the buttons
  - day names haev no border, no background, drawn with a grey color
  - days of the previous and next month have `LV_BTNMATRIX_CTRL_DISABLED` flag
  - today has a ticker border with the themes primary color
  - highlighted day have 40% opacity with the themes primary color.

Usage

Some funnctions uses the `lv_calendar_date_t` type is used which is a structure with `year`, `month` and `day` fields.

Current date

To set the current date (today), use the `lv_calendar_set_today_date(calendar, 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 arrays 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 array should be a static, global or constant variables.

Headers

Arrow buttons

`lv_calendar_header_arrow_create(parent, calendar, button_size)` creates a header that contains a left and right arrow on the sides and atext with the current year and month between them.

Dropdown

`lv_calendar_header_dropdown_create(parent, calendar)` creates a header that contains 2 drop-drown lists: one for the year and an other for the month.

Events

- `LV_EVENT_VALUE_CHANGED` Sent is a data is clicked. `lv_calendar_get_pressed_date(calendar, &date)` tells which day is currently being pressed. Returns `LV_RES_OK` if theres is valid pressed data, else `LV_RES_INV`.

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*.

Example

API

Functions

lv_obj_t ***lv_calendar_create**(*lv_obj_t* *parent)

void **lv_calendar_set_today_date**(*lv_obj_t* *obj, uint32_t year, uint32_t month, uint32_t day)

Set the today's date

Parameters

- **obj** -- pointer to a calendar object
- **year** -- today's year
- **month** -- today's month [1..12]
- **day** -- today's day [1..31]

void **lv_calendar_set_showed_date**(*lv_obj_t* *obj, uint32_t year, uint32_t month)

Set the currently showed

Parameters

- **obj** -- pointer to a calendar object
- **year** -- today's year
- **month** -- today's month [1..12]

void **lv_calendar_set_highlighted_dates**(*lv_obj_t* *obj, *lv_calendar_date_t* highlighted[], uint16_t date_num)

Set the the highlighted dates

Parameters

- **obj** -- pointer to a calendar object
- **highlighted** -- pointer to an `lv_calendar_date_t` array containing the dates. Only the pointer will be saved so this variable can't be local which will be destroyed later.
- **date_num** -- number of dates in the array

void **lv_calendar_set_day_names**(*lv_obj_t* *obj, const char **day_names)

Set the name of the days

Parameters

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

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

Get the today's date

Parameters **calendar** -- pointer to a calendar object

Returns return pointer to an `lv_calendar_date_t` variable containing the date of today.

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

Get the currently showed

Parameters **calendar** -- pointer to a calendar object

Returns pointer to an `lv_calendar_date_t` variable containing the date is being shown.

lv_calendar_date_t ***lv_calendar_get_highlighted_dates**(const *lv_obj_t* *calendar)
Get the the highlighted dates

Parameters **calendar** -- pointer to a calendar object

Returns pointer to an `lv_calendar_date_t` array containing the dates.

uint16_t **lv_calendar_get_highlighted_dates_num**(const *lv_obj_t* *calendar)
Get the number of the highlighted dates

Parameters **calendar** -- pointer to a calendar object

Returns number of highlighted days

lv_res_t **lv_calendar_get_pressed_date**(const *lv_obj_t* *calendar, *lv_calendar_date_t* *date)
Get the currently pressed day

Parameters

- **calendar** -- pointer to a calendar object
- **date** -- store the pressed date here

Returns LV_RES_OK: there is a valid pressed date; LV_RES_INV: there is no pressed data

Variables

const lv_obj_class_t **lv_calendar_class**

struct **lv_calendar_date_t**
#include <lv_calendar.h> Represents a date on the calendar object (platform-agnostic).

Public Members

uint16_t **year**

int8_t **month**

int8_t **day**
1..12

struct **lv_calendar_t**

Public Members

lv_btnmatrix_t **btnm**

lv_calendar_date_t **today**

lv_calendar_date_t **showed_date**

lv_calendar_date_t ***highlighted_dates**

uint16_t **highlighted_dates_num**

const char ***map**[8 * 7]

char **nums**[7 * 6][4]

5.3.2 Chart (lv_chart)

Overview

Charts are a basic object to visualize data points. They support *Line* charts (connect points with lines and/or draw points on them) and *Column* charts.

Charts also support division lines, 2 y axis, axis ticks, and texts on ticks.

Parts and Styles

The Chart's main part is called `LV_CHART_PART_BG` and it uses all the typical background properties. The *text* style properties determine the style of the axis texts and the *line* properties determine ticks' style. *Padding* values add some space on the sides thus it makes the *series area* smaller. Padding also can be used to make space for axis texts and ticks.

The background of the series is called `LV_CHART_PART_SERIES_BG` and it's placed on the main background. The division lines, and series data is drawn on this part. Besides the typical background style properties the *line* style properties are used by the division lines. The *padding* values tells the space between the this part and the axis texts.

The style of the series can be referenced by `LV_CHART_PART_SERIES`. In case of column type the following properties are used:

- *radius*: radius of the bars
- *padding_inner*: space between the columns of the same x coordinate

In case of Line type these properties are used:

- *line properties* to describe the lines
- *size* radius of the points
- *bg_opa*: the overall opacity of the area below the lines
- *bg_main_stop*: % of *bg_opa* at the top to create an alpha fade (0: transparent at the top, 255: *bg_opa* at the top)
- *bg_grad_stop*: % of *bg_opa* at the bottom to create an alpha fade (0: transparent at the bottom, 255: *bg_opa* at the top)
- *bg_drag_dir*: should be `LV_GRAD_DIR_VER` to allow alpha fading with *bg_main_stop* and *bg_grad_stop*

`LV_CHART_PART_CURSOR` refres to the cursors. Any number of cursor can be added and their appearence can be set by the line related style properties. The color of the cursors are set when the cursor is created and `line_color` fro mteh style is overwritten by this value.

Usage

Data series

You can add any number of series to the charts by `lv_chart_add_series(chart, color)`. It allocates data for a `lv_chart_series_t` structure which contains the chosen `color` and an array for the data points if not using an external array, if an external array is assigned any internal points associated with the series are deallocated and the series points to the external array instead.

Series' type

The following **data display types** exist:

- **LV_CHART_TYPE_NONE** - Do not display any data. It can be used to hide the series.
- **LV_CHART_TYPE_LINE** - Draw lines between the points.
- **LV_CHART_TYPE_COLUMN** - Draw columns.

You can specify the display type with `lv_chart_set_type(chart, LV_CHART_TYPE_...)`. The types can be 'OR'ed (like `LV_CHART_TYPE_LINE`).

Modify the data

You have several options to set the data of series:

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

Use `LV_CHART_POINT_DEF` as value to make the library skip drawing that point, column, or line segment.

Override default start point for series

If you wish a plot to start from a point other than the default which is point[0] of the series, you can set an alternative index with the function `lv_chart_set_x_start_point(chart, ser, id)` where id is the new index position to start plotting from.

Set an external data source

You can make the chart series update from an external data source by assigning it with the function: `lv_chart_set_ext_array(chart, ser, array, point_cnt )` where array is an external array of lv_coord_t with point_cnt elements. Note: you should call `lv_chart_refresh(chart)` after the external data source has been updated, to update the chart.

Get current chart information

There are four functions to get information about a chart:

1. `lv_chart_get_type(chart)` returns the current chart type.
2. `lv_chart_get_point_count(chart)` returns the current chart point count.
3. `lv_chart_get_x_start_point(ser)` returns the current plotting index for the specified series.
4. `lv_chart_get_point_id(chart, ser, id)` returns the value of the data at a particular index(id) for the specified series.

Update modes

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

- **LV_CHART_UPDATE_MODE_SHIFT** - Shift old data to the left and add the new one o the right.
- **LV_CHART_UPDATE_MODE_CIRCULAR** - Circularly add the new data (Like an ECG diagram).

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

Number of points

The number of points in the series can be modified by `lv_chart_set_point_count(chart, point_num)`. The default value is 10. Note: this also affects the number of points processed when an external buffer is assigned to a series.

Vertical range

You can specify the minimum and maximum values in y-direction with `lv_chart_set_range(chart, y_min, y_max)`. The value of the points will be scaled proportionally. The default range is: 0..100.

Division lines

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

Tick marks and labels

Ticks and labels can be added to the axis.

`lv_chart_set_x_tick_text(chart, list_of_values, num_tick_marks, LV_CHART_AXIS_...)` set the ticks and texts on x axis. `list_of_values` is a string with `'\n'` terminated text (expect the last) with text for the ticks. E.g. `const char * list_of_values = "first\nsec\nthird"`. `list_of_values` can be `NULL`. If `list_of_values` is set then `num_tick_marks` tells the number of ticks between two labels. If `list_of_values` is `NULL` then it specifies the total number of ticks.

Major tick lines are drawn where text is placed, and *minor tick lines* are drawn elsewhere. `lv_chart_set_x_tick_length(chart, major_tick_len, minor_tick_len)` sets the length of tick lines on the x-axis.

The same functions exists for the y axis too: `lv_chart_set_y_tick_text` and `lv_chart_set_y_tick_length`.

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_CHART_CURSOR_NONE/RIGHT/UP/LEFT/DOWN` or their OR-ed values to tell in which direction(s) should the cursor be drawn.

`lv_chart_set_cursor_point(chart, cursor, &point)` sets the position of the cursor. `point` is a pointer to an `lv_poin_t` variable. E.g. `lv_point_t point = {10, 20};`. The point is relative to the series area of the chart.

The `lv_coord_t p_index = lv_chart_get_nearest_index_from_coord(chart, x)` tells which point index is to the closest to a X coordinate (relative to the series area). It can be used to snap the cursor to a point for example when the chart is clicked.

`lv_chart_get_x_from_index(chart, series, id)` and `lv_chart_get_y_from_index(chart, series, id)` tells the X and Y coordinate of a given point. It's useful to place the cursor to given point.

The current series area can be retrieved with `lv_chart_get_series_area(chart, &area)` where `area` is a pointer to an `lv_area_t` variable to store the result. The area has absolute coordinates.

Events

Only the Generic events are sent by the object type.

Learn more about *Events*.

Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

Example

API

Typedefs

typedef uint8_t **lv_chart_type_t**

typedef uint8_t **lv_chart_update_mode_t**

typedef uint8_t **lv_chart_axis_t**

Enums

enum **[anonymous]**

Chart types

Values:

enumerator **LV_CHART_TYPE_NONE**

Don't draw the series

enumerator **LV_CHART_TYPE_LINE**

Connect the points with lines

enumerator **LV_CHART_TYPE_BAR**

Draw columns

enum **[anonymous]**

Chart update mode for `lv_chart_set_next`

Values:

enumerator **LV_CHART_UPDATE_MODE_SHIFT**

Shift old data to the left and add the new one the right

enumerator **LV_CHART_UPDATE_MODE_CIRCULAR**

Add the new data in a circular way

enum **[anonymous]**

Enumeration of the axis'

Values:

enumerator **LV_CHART_AXIS_PRIMARY_Y**

enumerator **LV_CHART_AXIS_SECONDARY_Y**

enumerator **LV_CHART_AXIS_X**

enumerator **_LV_CHART_AXIS_LAST**

Functions

LV_EXPORT_CONST_INT(LV_CHART_POINT_NONE)

lv_obj_t ***lv_chart_create**(*lv_obj_t* *parent)

Create a chart objects

Parameters **parent** -- pointer to an object, it will be the parent of the new button

Returns pointer to the created chart

void **lv_chart_set_type**(*lv_obj_t* *obj, *lv_chart_type_t* type)

Set a new type for a chart

Parameters

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

void **lv_chart_set_point_count**(*lv_obj_t* *obj, uint16_t cnt)

Set the number of points on a data line on a chart

Parameters

- **obj** -- pointer r to chart object
- **cnt** -- new number of points on the data lines

void **lv_chart_set_range**(*lv_obj_t* *obj, *lv_chart_axis_t* axis, lv_coord_t min, lv_coord_t max)

Set the minimal and maximal y values on an axis

Parameters

- **obj** -- pointer to a chart object
- **axis** -- LV_CHART_AXIS_PRIMARY_Y or LV_CHART_AXIS_SECONDARY_Y
- **min** -- minimum value of the y axis
- **max** -- maximum value of the y axis

void **lv_chart_set_update_mode**(*lv_obj_t* *obj, *lv_chart_update_mode_t* update_mode)

Set update mode of the chart object. Affects

Parameters

- **obj** -- pointer to a chart object
- **mode** -- the update mode

void **lv_chart_set_div_line_count**(*lv_obj_t* *obj, uint8_t hdiv, uint8_t vdiv)

Set the number of horizontal and vertical division lines

Parameters

- **obj** -- pointer to a chart object
- **hdiv** -- number of horizontal division lines
- **vdiv** -- number of vertical division lines

void **lv_chart_set_zoom_x**(*lv_obj_t* *obj, uint16_t zoom_x)

Zoom into the chart in X direction

Parameters

- **obj** -- pointer to a chart object
- **zoom_x** -- zoom in x direction. LV_ZOOM_NONE or 256 for no zoom, 512 double zoom

void **lv_chart_set_zoom_y**(*lv_obj_t* *obj, uint16_t zoom_y)

Zoom into the chart in Y direction

Parameters

- **obj** -- pointer to a chart object
- **zoom_y** -- zoom in y direction. LV_ZOOM_NONE or 256 for no zoom, 512 double zoom

uint16_t **lv_chart_get_zoom_x**(const *lv_obj_t* *obj)

Get X zoom of a chart

Parameters **obj** -- pointer to a chart object

Returns the X zoom value

uint16_t **lv_chart_get_zoom_y**(const *lv_obj_t* *obj)

Get Y zoom of a chart

Parameters **obj** -- pointer to a chart object

Returns the Y zoom value

void **lv_chart_set_axis_tick**(*lv_obj_t* *obj, *lv_chart_axis_t* axis, lv_coord_t major_len, lv_coord_t minor_len, lv_coord_t major_cnt, lv_coord_t minor_cnt, bool label_en, lv_coord_t draw_size)

Set the number of tick lines on an axis

Parameters

- **obj** -- pointer to a chart object
- **axis** -- an axis which ticks count should be set
- **major_len** -- length of major ticks
- **minor_len** -- length of minor ticks
- **major_cnt** -- number of major ticks on the axis
- **minor_cnt** -- 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)

lv_chart_type_t **lv_chart_get_type**(const *lv_obj_t* *obj)

Get the type of a chart

Parameters **obj** -- pointer to chart object

Returns type of the chart (from '*lv_chart_t*' enum)

uint16_t **lv_chart_get_point_count**(const *lv_obj_t* *obj)

Get the data point number per data line on chart

Parameters **chart** -- pointer to chart object

Returns point number on each data line

uint16_t **lv_chart_get_x_start_point**(const *lv_obj_t* *obj, *lv_chart_series_t* *ser)

Get the current index of the x-axis start point in the data array

Parameters

- **chart** -- pointer to a chart object
- **ser** -- pointer to a data series on 'chart'

Returns the index of the current x start point in the data array

void **lv_chart_get_point_pos_by_id**(*lv_obj_t* *obj, *lv_chart_series_t* *ser, uint16_t id, lv_point_t *p_out)

Get the position of point of the an index relative to the chart.

Parameters

- **chart** -- pointer to a chart object
- **ser** -- pointer to series
- **id** -- the index.
- **p_out** -- store the result position here

void **lv_chart_refresh**(*lv_obj_t* *obj)

Refresh a chart if its data line has changed

Parameters **chart** -- pointer to chart object

lv_chart_series_t ***lv_chart_add_series**(*lv_obj_t* *obj, lv_color_t color, *lv_chart_axis_t* axis)

Allocate and add a data series to the chart

Parameters

- **obj** -- pointer to a chart object
- **color** -- color of the data series
- **axis** -- the y axis to which the series should be attached (::LV_CHART_AXIS_PRIMARY_Y or ::LV_CHART_AXIS_SECONDARY_Y)

Returns pointer to the allocated data series

void **lv_chart_remove_series**(*lv_obj_t* *obj, *lv_chart_series_t* *series)

Deallocate and remove a data series from a chart

Parameters

- **chart** -- pointer to a chart object
- **series** -- pointer to a data series on 'chart'

void **lv_chart_hide_series**(*lv_obj_t* *chart, *lv_chart_series_t* *series, bool hide)

Hide/Unhide a single series of a chart.

Parameters

- **obj** -- pointer to a chart object.
- **series** -- pointer to a series object
- **hide** -- true: hide the series

void **lv_chart_set_series_color**(*lv_obj_t* *chart, *lv_chart_series_t* *series, lv_color_t color)

Change the color of a series

Parameters

- **obj** -- pointer to a chart object.
- **series** -- pointer to a series object

- **color** -- the new color of the series

void **lv_chart_set_x_start_point**(*lv_obj_t* *obj, *lv_chart_series_t* *ser, uint16_t id)

Set the index of the x-axis start point in the data array. This point will be considers the first (left) point and the other points will be drawn after it.

Parameters

- **obj** -- pointer to a chart object
- **ser** -- pointer to a data series on 'chart'
- **id** -- the index of the x point in the data array

lv_chart_series_t ***lv_chart_get_series_next**(const *lv_obj_t* *chart, const *lv_chart_series_t* *ser)

Get the next series.

Parameters

- **chart** -- pointer to a chart
- **ser** -- the previous series or NULL to get the first

Returns the next series or NULL if thre is no more.

lv_chart_cursor_t ***lv_chart_add_cursor**(*lv_obj_t* *obj, lv_color_t color, lv_dir_t dir)

Add a cursor with a given color

Parameters

- **obj** -- pointer to chart object
- **color** -- color of the cursor
- **dir** -- direction of the cursor. `LV_DIR_RIGHT/LEFT/TOP/DOWN/HOR/VER/ALL`. OR-ed values are possible

Returns pointer to the created cursor

void **lv_chart_set_cursor_point**(*lv_obj_t* *chart, *lv_chart_cursor_t* *cursor, lv_point_t *point)

Set the coordinate of the cursor with respect to the paddings

Parameters

- **obj** -- pointer to a chart object
- **cursor** -- pointer to the cursor
- **point** -- the new coordinate of cursor relative to paddings of the background

lv_point_t **lv_chart_get_cursor_point**(*lv_obj_t* *chart, *lv_chart_cursor_t* *cursor)

Get the coordinate of the cursor with respect to the paddings

Parameters

- **obj** -- pointer to a chart object
- **cursor** -- pointer to cursor

Returns coordinate of the cursor as lv_point_t

void **lv_chart_set_all_value**(*lv_obj_t* *obj, *lv_chart_series_t* *ser, lv_coord_t value)

Initialize all data points of a series with a value

Parameters

- **obj** -- pointer to chart object
- **ser** -- pointer to a data series on 'chart'

- **value** -- the new value for all points. `LV_CHART_POINT_DEF` can be used to hide the points.

void **lv_chart_set_next_value**(*lv_obj_t* *obj, *lv_chart_series_t* *ser, lv_coord_t value)

Set the next point according to the update mode policy.

Parameters

- **obj** -- pointer to chart object
- **ser** -- pointer to a data series on 'chart'
- **value** -- the new value of the next data

void **lv_chart_set_value_by_id**(*lv_obj_t* *obj, *lv_chart_series_t* *ser, lv_coord_t value, uint16_t id)

Set an individual point's y value of a chart's series directly based on its index

Parameters

- **obj** -- pointer to a chart object
- **ser** -- pointer to a data series on 'chart'
- **value** -- value to assign to array point
- **id** -- the index of the x point in the array

void **lv_chart_set_ext_array**(*lv_obj_t* *obj, *lv_chart_series_t* *ser, lv_coord_t array[])

Set an external array of data points to use for the chart NOTE: It is the users responsibility to make sure the `point_cnt` matches the external array size.

Parameters

- **obj** -- pointer to a chart object
- **ser** -- pointer to a data series on 'chart'
- **array** -- external array of points for chart

lv_coord_t ***lv_chart_get_array**(const *lv_obj_t* *obj, *lv_chart_series_t* *ser)

Get the array of values of a series

Parameters

- **obj** -- pointer to a chart object
- **ser** -- pointer to a data series on 'chart'

Returns the array of values with 'point_count' elements

uint32_t **lv_chart_get_pressed_point**(const *lv_obj_t* *obj)

Get the index of the currently pressed point. It's the same for every series.

Parameters **obj** -- pointer to a chart object

Returns the index of the point [0 .. point count] or LV_CHART_POINT_ID_NONE if no point is being pressed

Variables

const lv_obj_class_t **lv_chart_class**

struct **lv_chart_series_t**

#include <lv_chart.h> Descriptor a chart series

Public Members

lv_coord_t ***points**

lv_color_t **color**

uint16_t **last_point**

uint8_t **hidden**

uint8_t **ext_buf_assigned**

lv_chart_axis_t **y_axis**

struct **lv_chart_cursor_t**

Public Members

lv_point_t **point**

lv_color_t **color**

lv_dir_t **dir**

struct **lv_chart_tick_dsc_t**

Public Members

lv_coord_t **major_len**

lv_coord_t **minor_len**

lv_coord_t **draw_size**

uint32_t **minor_cnt**

uint32_t **major_cnt**

uint32_t **label_en**

struct **lv_chart_t**

Public Members

lv_obj_t **obj**

lv_ll_t **series_ll**
Linked list for the series (stores *lv_chart_series_t*)

lv_ll_t **cursor_ll**
Linked list for the cursors (stores *lv_chart_cursor_t*)

lv_chart_tick_dsc_t **tick**[*_LV_CHART_AXIS_LAST*]

lv_coord_t **ymin**[2]

lv_coord_t **ymax**[2]

uint16_t **pressed_point_id**

uint16_t **hdiv_cnt**
Number of horizontal division lines

uint16_t **vdiv_cnt**
Number of vertical division lines

uint16_t **point_cnt**
Point number in a data line

uint16_t **zoom_x**

uint16_t **zoom_y**

lv_chart_type_t **type**
Line or column chart

lv_chart_update_mode_t **update_mode**

5.3.3 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. Before reading this section, please read the Button section for better understanding.

Parts and Styles

The Image button object has only a main part called `LV_IMG_BTN_PART_MAIN` from where all *image* style properties are used. It's possible to recolor the image in each state with *image_recolor* and *image_recolor_opa* proeprties. For example, to make the image darker if it is pressed.

Usage

Image sources

To set the image in a state, use the `lv_imgbtn_set_src(imgbtn, LV_BTN_STATE_..., &img_src)`. The image sources works the same as described in the Image object except that, "Symbols" are not supported by the Image button.

If `LV_IMGBTN_TILED` is enabled in *lv_conf.h*, then `lv_imgbtn_set_src_tiled(imgbtn, LV_BTN_STATE_..., &img_src_left, &img_src_mid, &img_src_right)` becomes available. Using the tiled feature the *middle* image will be repeated to fill the width of the object. Therefore with `LV_IMGBTN_TILED`, you can set the width of the Image button using `lv_obj_set_width()`. However, without this option, the width will be always the same as the image source's width.

Button features

Similarly to normal Buttons `lv_imgbtn_set_checkable(imgbtn, true/false)`, `lv_imgbtn_toggle(imgbtn)` and `lv_imgbtn_set_state(imgbtn, LV_BTN_STATE_...)` also works.

Events

Beside the Generic events, the following Special events are sent by the buttons:

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

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

Learn more about *Events*.

Keys

The following *Keys* are processed by the Buttons:

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

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

Learn more about *Keys*.

Example

API

Enums

enum **lv_imgbtn_state_t**

Values:

enumerator **LV_IMGBTN_STATE_RELEASED**

enumerator **LV_IMGBTN_STATE_PRESSED**

enumerator **LV_IMGBTN_STATE_DISABLED**

enumerator **LV_IMGBTN_STATE_CHECKED_RELEASED**

enumerator **LV_IMGBTN_STATE_CHECKED_PRESSED**

enumerator **LV_IMGBTN_STATE_CHECKED_DISABLED**

enumerator **_LV_IMGBTN_STATE_NUM**

Functions

lv_obj_t ***lv_imgbtn_create**(*lv_obj_t* *parent)

Create a image button objects

Parameters **par** -- pointer to an object, it will be the parent of the new image button

Returns pointer to the created image button

void **lv_imgbtn_set_src**(*lv_obj_t* *imgbtn, *lv_imgbtn_state_t* state, const void *src_left, const void *src_mid, const void *src_right)

Set images for a state of the image button

Parameters

- **imgbtn** -- pointer to an image button object
- **state** -- for which state set the new image
- **src_left** -- pointer to an image source for the left side of the button (a C array or path to a file)
- **src_mid** -- pointer to an image source for the middle of the button (ideally 1px wide) (a C array or path to a file)
- **src_right** -- pointer to an image source for the right side of the button (a C array or path to a file)

const void ***lv_imgbtn_get_src_left**(*lv_obj_t* *imgbtn, *lv_imgbtn_state_t* state)

Get the left image in a given state

Parameters

- **imgbtn** -- pointer to an image button object
- **state** -- the state where to get the image (from `lv_btn_state_t`)`

Returns pointer to the left image source (a C array or path to a file)

const void *`lv_imgbtn_get_src_middle`(*lv_obj_t* *imgbtn, *lv_imgbtn_state_t* state)

Get the middle image in a given state

Parameters

- **imgbtn** -- pointer to an image button object
- **state** -- the state where to get the image (from `lv_btn_state_t`)`

Returns pointer to the middle image source (a C array or path to a file)

const void *`lv_imgbtn_get_src_right`(*lv_obj_t* *imgbtn, *lv_imgbtn_state_t* state)

Get the right image in a given state

Parameters

- **imgbtn** -- pointer to an image button object
- **state** -- the state where to get the image (from `lv_btn_state_t`)`

Returns pointer to the left image source (a C array or path to a file)

Variables

const lv_obj_class_t `lv_imgbtn_class`

struct `lv_imgbtn_t`

Public Members

lv_obj_t `obj`

const void *`img_src_mid`[*_LV_IMGBTN_STATE_NUM*]

const void *`img_src_left`[*_LV_IMGBTN_STATE_NUM*]

const void *`img_src_right`[*_LV_IMGBTN_STATE_NUM*]

lv_img_cf_t `act_cf`

5.3.4 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 text.

Parts and Styles

Similarly to Button matices Keyboards consist of 2 part:

- `LV_KEYBOARD_PART_BG` which is the main part and uses all the typical background properties
- `LV_KEYBOARD_PART_BTN` which is virtual part for the buttons. It also uses all typical backround proeprties and the *text* properties.

Usage

Modes

The Keyboards have thefollowing modes:

- **LV_KEYBOARD_MODE_TEXT_LOWER** - Display lower case letters
- **LV_KEYBOARD_MODE_TEXT_UPPER** - Display upper case letters
- **LV_KEYBOARD_MODE_TEXT_SPECIAL** - Display special characters
- **LV_KEYBOARD_MODE_NUM** - Display numbers, +/- sign, and decimal dot.

The `TEXT` modes' layout contains buttons to change mode.

To set the mode manually, use `lv_keyboard_set_mode(kb, mode)`. The default more is `LV_KEYBOARD_MODE_TEXT_UPPER`.

Assign Text area

You can assign a Text area to the Keyboard to automatically put the clicked characters there. To assign the text area, use `lv_keyboard_set_textarea(kb, ta)`.

The assigned text area's **cursor can be managed** by the keyboard: when the keyboard is assigned, the previous text area's cursor will be hidden and the new one will be shown. When the keyboard is closed by the *Ok* or *Close* buttons, the cursor also will be hidden. The cursor manager feature is enabled by `lv_keyboard_set_cursor_manage(kb, true)`. The default is not managed.

New Keymap

You can specify a new map (layout) for the keyboard with `lv_keyboard_set_map(kb, map)` and `lv_keyboard_set_ctrl_map(kb, ctrl_map)`. Learn more about the Button matrix object. Keep in mind that, using following keywords will have the same effect as with the original map:

- *LV_SYMBOL_OK* - Apply.
- *LV_SYMBOL_CLOSE* - Close.
- *LV_SYMBOL_BACKSPACE* - Delete on the left.
- *LV_SYMBOL_LEFT* - Move the cursor left.
- *LV_SYMBOL_RIGHT* - Move the cursor right.
- *"ABC"* - Load the uppercase map.
- *"abc"* - Load the lower case map.
- *"Enter"* - New line.

Events

Besides the Generic events, the following Special events are sent by the keyboards:

- **LV_EVENT_VALUE_CHANGED** - Sent when the button is pressed/released or repeated after long press. The event data is set to the ID of the pressed/released button.
- **LV_EVENT_APPLY** - The *Ok* button is clicked.
- **LV_EVENT_CANCEL** - The *Close* button is clicked.

The keyboard has a **default event handler** callback called `lv_keyboard_def_event_cb`. It handles the button pressing, map changing, the assigned text area, etc. You can completely replace it with your custom event handler however, you can call `lv_keyboard_def_event_cb` at the beginning of your event handler to handle the same things as before.

Learn more about *Events*.

Keys

The following *Keys* are processed by the buttons:

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

Learn more about *Keys*.

Examples

API

Typedefs

typedef uint8_t **`lv_keyboard_mode_t`**

Enums

enum **`[anonymous]`**

Current keyboard mode.

Values:

enumerator **`LV_KEYBOARD_MODE_TEXT_LOWER`**

enumerator **`LV_KEYBOARD_MODE_TEXT_UPPER`**

enumerator **`LV_KEYBOARD_MODE_SPECIAL`**

enumerator **`LV_KEYBOARD_MODE_NUMBER`**

Functions

lv_obj_t ***lv_keyboard_create**(*lv_obj_t* *parent)

Create a keyboard objects

Parameters **par** -- pointer to an object, it will be the parent of the new keyboard

Returns pointer to the created keyboard

void **lv_keyboard_set_textarea**(*lv_obj_t* *kb, *lv_obj_t* *ta)

Assign a Text Area to the Keyboard. The pressed characters will be put there.

Parameters

- **kb** -- pointer to a Keyboard object
- **ta** -- pointer to a Text Area object to write there

void **lv_keyboard_set_mode**(*lv_obj_t* *kb, *lv_keyboard_mode_t* mode)

Set a new a mode (text or number map)

Parameters

- **kb** -- pointer to a Keyboard object
- **mode** -- the mode from 'lv_keyboard_mode_t'

void **lv_keyboard_set_map**(*lv_obj_t* *kb, *lv_keyboard_mode_t* mode, const char *map[], const *lv_btnmatrix_ctrl_t* ctrl_map[])

Set a new map for the keyboard

Parameters

- **kb** -- pointer to a Keyboard object
- **mode** -- keyboard map to alter 'lv_keyboard_mode_t'
- **map** -- pointer to a string array to describe the map. See 'lv_btnmatrix_set_map()' for more info.

lv_obj_t ***lv_keyboard_get_textarea**(const *lv_obj_t* *kb)

Assign a Text Area to the Keyboard. The pressed characters will be put there.

Parameters **kb** -- pointer to a Keyboard object

Returns pointer to the assigned Text Area object

lv_keyboard_mode_t **lv_keyboard_get_mode**(const *lv_obj_t* *kb)

Set a new a mode (text or number map)

Parameters **kb** -- pointer to a Keyboard object

Returns the current mode from 'lv_keyboard_mode_t'

static inline const char ****lv_keyboard_get_map_array**(const *lv_obj_t* *kb)

Get the current map of a keyboard

Parameters **kb** -- pointer to a keyboard object

Returns the current map

void **lv_keyboard_def_event_cb**(lv_event_t *e)

Default keyboard event to add characters to the Text area and change the map. If a custom `event_cb` is added to the keyboard this function be called from it to handle the button clicks

Parameters

- **kb** -- pointer to a keyboard
- **event** -- the triggering event

Variables

const lv_obj_class_t **lv_keyboard_class**

struct **lv_keyboard_t**

Public Members

lv_btnmatrix_t **btnm**

lv_obj_t ***ta**

lv_keyboard_mode_t **mode**

5.3.5 LED (lv_led)

Overview

The LEDs are rectangle-like (or circle) object. It's brightness can be adjusted. With lower brightness the 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

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

Only the Generic events are sent by the object type.

Learn more about *Events*.

Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

Example

API

Functions

lv_obj_t ***lv_led_create**(*lv_obj_t* *parent)
Create a led objects

Parameters **par** -- pointer to an object, it will be the parent of the new led

Returns pointer to the created led

void **lv_led_set_color**(*lv_obj_t* *led, lv_color_t color)
Set the color of the LED

Parameters

- **led** -- pointer to a LED object
- **color** -- the color of the the LED

void **lv_led_set_brightness**(*lv_obj_t* *led, uint8_t bright)
Set the brightness of a LED object

Parameters

- **led** -- pointer to a LED object
- **bright** -- LV_LED_BRIGHT_MIN (max. dark) ... LV_LED_BRIGHT_MAX (max. light)

void **lv_led_on**(*lv_obj_t* *led)
Light on a LED

Parameters **led** -- pointer to a LED object

void **lv_led_off**(*lv_obj_t* *led)
Light off a LED

Parameters **led** -- pointer to a LED object

void **lv_led_toggle**(*lv_obj_t* *led)
Toggle the state of a LED

Parameters **led** -- pointer to a LED object

uint8_t **lv_led_get_brightness**(const *lv_obj_t* *obj)
Get the brightness of a LEd object

Parameters **led** -- pointer to LED object

Returns bright 0 (max. dark) ... 255 (max. light)

Variables

const lv_obj_class_t **lv_led_class**

struct **lv_led_t**

Public Members

lv_obj_t **obj**

lv_color_t **color**

uint8_t **bright**
Current brightness of the LED (0..255)

5.3.6 List (lv_list)

Overview

The Lists are built from a background Page and Buttons on it. The Buttons contain an optional icon-like Image (which can be a symbol too) and a Label. When the list becomes long enough it can be scrolled.

Parts and Styles

The List has the same parts as the Page

- `LV_LIST_PART_BG`
- `LV_LIST_PART_SCRL`
- `LV_LIST_PART_SCRLBAR`
- `LV_LIST_PART_EDGE_FLASH`

Refer to the Page documentation for details.

The buttons on the list are treated as normal buttons and they only have a main part called `LV_BTN_PART_MAIN`.

Usage

Add buttons

You can add new list elements (button) with `lv_list_add_btn(list, &icon_img, "Text")` or with symbol `lv_list_add_btn(list, SYMBOL_EDIT, "Edit text")`. If you do not want to add image use `NULL` as image source. The function returns with a pointer to the created button to allow further configurations.

The width of the buttons is set to maximum according to the object width. The height of the buttons are adjusted automatically according to the content. (*content height* + *padding_top* + *padding_bottom*).

The labels are created with `LV_LABEL_LONG_SROLL_CIRC` long mode to automatically scroll the long labels circularly.

`lv_list_get_btn_label(list_btn)` and `lv_list_get_btn_img(list_btn)` can be used to get the label and the image of a list button. The text can be et directly with `lv_list_get_btn_text(list_btn)`.

Delete buttons

To delete a list element use `lv_list_remove(list, btn_index)`. btn_index can be obtained by `lv_list_get_btn_index(list, btn)` where btn is the return value of `lv_list_add_btn()`.

To clean the list (remove all buttons) use `lv_list_clean(list)`

Manual navigation

You can navigate manually in the list with `lv_list_up(list)` and `lv_list_down(list)`.

You can focus on a button directly using `lv_list_focus(btn, LV_ANIM_ON/OFF)`.

The **animation time** of up/down/focus movements can be set via: `lv_list_set_anim_time(list, anim_time)`. Zero animation time means not animations.

Layout

By default the list is vertical. To get a horizontal list use `lv_list_set_layout(list, LV_LAYOUT_ROW_MID)`.

Edge flash

A circle-like effect can be shown when the list reaches the most top or bottom position. `lv_list_set_edge_flash(list, true)` enables this feature.

Scroll propagation

If the list is created on an other scrollable element (like a Page) and the list can't be scrolled further the scrolling can be propagated to the parent. This way the scroll will be continued on the parent. It can be enabled with `lv_list_set_scroll_propagation(list, true)`

Events

Only the Generic events are sent by the object type.

Learn more about *Events*.

Keys

The following *Keys* are processed by the Lists:

- **LV_KEY_RIGHT/DOWN** Select the next button
- **LV_KEY_LEFT/UP** Select the previous button

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

The Selected buttons are in `LV_BTN_STATE_PR/TG_PR` state.

To manually select a button use `lv_list_set_btn_selected(list, btn)`. When the list is defocused and focused again it will restore the last selected button.

Learn more about *Keys*.

Example

API

Functions

lv_obj_t ***lv_list_create**(*lv_obj_t* *parent)

lv_obj_t ***lv_list_add_text**(*lv_obj_t* *list, const char *txt)

lv_obj_t ***lv_list_add_btn**(*lv_obj_t* *list, const char *icon, const char *txt)

const char ***lv_list_get_btn_text**(*lv_obj_t* *list, *lv_obj_t* *btn)

Variables

const lv_obj_class_t **lv_list_class**

const lv_obj_class_t **lv_list_text_class**

const lv_obj_class_t **lv_list_btn_class**

5.3.7 Message box (lv_msgbox)

Overview

The Message boxes act as pop-ups. They are built from a background Container, a Label and a Button matrix for buttons.

The text will be broken into multiple lines automatically (has `LV_LABEL_LONG_MODE_BREAK`) and the height will be set automatically to involve the text and the buttons (`LV_FIT_TIGHT` fit vertically)-

Parts and Styles

The Message box's main part is called `LV_MSGBOX_PART_MAIN` and it uses all the typical background style properties. Using padding will add space on the sides. *pad_inner* will add space between the text and the buttons. The *label* style properties affect the style of text.

The buttons parts are the same as in case of Button matrix:

- `LV_MSGBOX_PART_BTN_BG` the background of the buttons
- `LV_MSGBOX_PART_BTN` the buttons

Usage

Set text

To set the text use the `lv_msgbox_set_text(msgbox, "My text")` function. Not only the pointer of the text will be saved, so the the text can be in a local variable too.

Add buttons

To add buttons use the `lv_msgbox_add_btns(msgbox, btn_str)` function. The button's text needs to be specified like `const char * btn_str[] = {"Apply", "Close", ""}`. For more information visit the Button matrix documentation.

The button matrix will be created only when `lv_msgbox_add_btns()` is called for the first time.

Auto-close

With `lv_msgbox_start_auto_close(mbox, delay)` the message box can be closed automatically after `delay` milliseconds with an animation. The `lv_mbox_stop_auto_close(mbox)` function stops a started auto close.

The duration of the close animation can be set by `lv_mbox_set_anim_time(mbox, anim_time)`.

Events

Besides the Generic events the following Special events are sent by the Message boxes:

- **LV_EVENT_VALUE_CHANGED** sent when the button is clicked. The event data is set to ID of the clicked button.

The Message box has a default event callback which closes itself when a button is clicked.

Learn more about *Events*.

##Keys

The following *Keys* are processed by the Buttons:

- **LV_KEY_RIGHT/DOWN** Select the next button
- **LV_KEY_LEFT/TOP** Select the previous button
- **LV_KEY_ENTER** Clicks the selected button

Learn more about *Keys*.

Example

API

Functions

lv_obj_t ***lv_msgbox_create**(*lv_obj_t* *parent, const char *title, const char *txt, const char *btn_txts[], bool add_close_btn)

Create a message box objects

Parameters

- **parent** -- pointer to parent or NULL to create a full screen modal message box
- **title** -- the title of the message box
- **txt** -- the text of the message box
- **btn_txts** -- the buttons as an array of texts terminated by an "" element. E.g. {"btn1", "btn2", ""}
- **add_close_btn** -- true: add a close button

Returns pointer to the message box object

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)

const char ***lv_msgbox_get_active_btn_text**(*lv_obj_t* *mbox)

void **lv_msgbox_close**(*lv_obj_t* *mbox)

Variables

const lv_obj_class_t **lv_msgbox_class**

5.3.8 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 Spinbox's main part is called `LV_SPINBOX_PART_BG` which is a rectangle-like background using all the typical background style properties. It also describes the style of the label with its *text* style properties.

`LV_SPINBOX_PART_CURSOR` is a virtual part describing the cursor. Read the Text area documentation for a detailed description.

Set format

`lv_spinbox_set_digit_format(spinbox, digit_count, separator_position)` set the format of the number. `digit_count` sets the number of digits. Leading zeros are added to fill the space on the left. `separator_position` sets the number of digit before the decimal point. `0` means no decimal point.

`lv_spinbox_set_padding_left(spinbox, cnt)` add `cnt` "space" characters between the sign an the most left digit.

Value and ranges

`lv_spinbox_set_range(spinbox, min, max)` sets the range of the Spinbox.

`lv_spinbox_set_value(spinbox, num)` sets the Spinbox's value manually.

`lv_spinbox_increment(spinbox)` and `lv_spinbox_decrement(spinbox)` increments/decrements the value of the Spinbox.

`lv_spinbox_set_step(spinbox, step)` sets the amount to increment decrement.

Events

Besides the Generic events the following Special events are sent by the Drop down lists:

- **LV_EVENT_VALUE_CHANGED** sent when the value has changed. (the value is set as event data as `int32_t`)
- **LV_EVENT_INSERT** sent by the ancestor Text area but shouldn't be used.

Learn more about *Events*.

Keys

The following *Keys* are processed by the Buttons:

- **LV_KEY_LEFT/RIGHT** With *Keypad* move the cursor left/right. With *Encoder* decrement/increment the selected digit.
- **LY_KEY_ENTER** Apply the selected option (Send `LV_EVENT_VALUE_CHANGED` event and close the Drop down list)
- **LV_KEY_ENTER** With *Encoder* got the net digit. Jump to the first after the last.

Example

API

Functions

lv_obj_t ***lv_spinbox_create**(*lv_obj_t* *parent)
Create a spinbox objects

Parameters **par** -- pointer to an object, it will be the parent of the new spinbox

Returns pointer to the created spinbox

void **lv_spinbox_set_rollover**(*lv_obj_t* *obj, bool b)
Set spinbox rollover function

Parameters

- **spinbox** -- pointer to spinbox
- **b** -- true or false to enable or disable (default)

void **lv_spinbox_set_value**(*lv_obj_t* *obj, int32_t i)
Set spinbox value

Parameters

- **spinbox** -- pointer to spinbox
- **i** -- value to be set

void **lv_spinbox_set_digit_format**(*lv_obj_t* *obj, uint8_t digit_count, uint8_t separator_position)
Set spinbox digit format (digit count and decimal format)

Parameters

- **spinbox** -- pointer to spinbox
- **digit_count** -- number of digit excluding the decimal separator and the sign
- **separator_position** -- number of digit before the decimal point. If 0, decimal point is not shown

void **lv_spinbox_set_step**(*lv_obj_t* *obj, uint32_t step)
Set spinbox step

Parameters

- **spinbox** -- pointer to spinbox
- **step** -- steps on increment/decrement

void **lv_spinbox_set_range**(*lv_obj_t* *obj, int32_t range_min, int32_t range_max)
Set spinbox value range

Parameters

- **spinbox** -- pointer to spinbox
- **range_min** -- maximum value, inclusive
- **range_max** -- minimum value, inclusive

bool **lv_spinbox_get_rollover**(*lv_obj_t* *obj)
Get spinbox rollover function status

Parameters **spinbox** -- pointer to spinbox

int32_t **lv_spinbox_get_value**(*lv_obj_t* *obj)
Get the spinbox numeral value (user has to convert to float according to its digit format)

Parameters **spinbox** -- pointer to spinbox

Returns value integer value of the spinbox

int32_t **lv_spinbox_get_step**(*lv_obj_t* *obj)
Get the spinbox step value (user has to convert to float according to its digit format)

Parameters **spinbox** -- pointer to spinbox

Returns value integer step value of the spinbox

void **lv_spinbox_step_next**(*lv_obj_t* *obj)
Select next lower digit for edition by dividing the step by 10

Parameters **spinbox** -- pointer to spinbox

void **lv_spinbox_step_prev**(*lv_obj_t* *obj)
Select next higher digit for edition by multiplying the step by 10

Parameters **spinbox** -- pointer to spinbox

void **lv_spinbox_increment**(*lv_obj_t* *obj)
Increment spinbox value by one step

Parameters **spinbox** -- pointer to spinbox

void **lv_spinbox_decrement**(*lv_obj_t* *obj)
Decrement spinbox value by one step

Parameters **spinbox** -- pointer to spinbox

Variables

const lv_obj_class_t **lv_spinbox_class**

struct **lv_spinbox_t**

Public Members

lv_textarea_t **ta**

int32_t **value**

int32_t **range_max**

int32_t **range_min**

int32_t **step**

uint16_t **digit_count**

uint16_t **dec_point_pos**

uint16_t **rollover**

Example

5.3.9 Spinner (lv_spinner)

Overview

The Spinner object is a spinning arc over a border.

Parts and Styles

The Spinner uses the the following parts:

- `LV_SPINNER_PART_BG`: main part
- `LV_SPINNER_PART_INDIC`: the spinning arc (virtual part)

The parts and style works the same as in case of Arc. Read its documentation for a details description.

Usage

Arc length

The length of the arc can be adjusted by `lv_spinner_set_arc_length(spinner, deg)`.

Spinning speed

The speed of the spinning can be adjusted by `lv_spinner_set_spin_time(preload, time_ms)`.

Spin types

You can choose from more spin types:

- **LV_SPINNER_TYPE_SPINNING_ARC** spin the arc, slow down on the top
- **LV_SPINNER_TYPE_FILLSPIN_ARC** spin the arc, slow down on the top but also stretch the arc
- **LV_SPINNER_TYPE_CONSTANT_ARC** spin the arc at a constant speed

To apply one if them use `lv_spinner_set_type(preload, LV_SPINNER_TYPE_...)`

Spin direction

The direction of spinning can be changed with `lv_spinner_set_dir(preload, LV_SPINNER_DIR_FORWARD/BACKWARD)`.

Events

Only the Generic events are sent by the object type.

Keys

No *Keys* are processed by the object type.

Learn more about *Keys*.

Example

MicroPython

No examples yet.

API

Functions

lv_obj_t ***lv_spinner_create**(*lv_obj_t* *parent, uint32_t time, uint32_t arc_length)

5.3.10 Tabview (lv_tabview)

Overview

The Tab view object can be used to organize content in tabs.

Parts and Styles

The Tab view object has several parts. The main is `LV_TABVIEW_PART_BG`. It a rectangle-like container which holds the other parts of the Tab view.

On the background 2 important real parts are created:

- `LV_TABVIEW_PART_BG_SCRL`: it's the scrollable part of Page. It holds the content of the tabs next to each other. The background of the Page is always transparent and can't be accessed externally.
- `LV_TABVIEW_PART_TAB_BG`: The tab buttons which is a Button matrix. Clicking on a button will scroll `LV_TABVIEW_PART_BG_SCRL` to the related tab's content. The tab buttons can be accessed via `LV_TABVIEW_PART_TAB_BTN`. When tabs are selected, the buttons are in the checked state, and can be styled using `LV_STATE_CHECKED`. The height of the tab's button matrix is calculated from the font height plus padding of the background's and the button's style.

All the listed parts supports the typical background style properties and padding.

`LV_TABVIEW_PART_TAB_BG` has an additional real part, an indicator, called `LV_TABVIEW_PART_INDIC`. It's a thin rectangle-like object under the currently selected tab. When the tab view is animated to an other tab the indicator will be animated too. It can be styles using the typical background style properties. The *size* style property will set the its thickness.

When a new tab is added a Page is create for them on `LV_TABVIEW_PART_BG_SCRL` and a new button is added to `LV_TABVIEW_PART_TAB_BG` Button matrix. The created Pages can be used as normal Pages and they have the usual Page parts.

Usage

Adding tab

New tabs can be added with `lv_tabview_add_tab(tabview, "Tab name")`. It will return with a pointer to a Page object where the tab's content can be created.

Change tab

To select a new tab you can:

- Click on it on the Button matrix part
- Slide
- Use `lv_tabview_set_tab_act(tabview, id, LV_ANIM_ON/OFF)` function

Change tab's name

To change the name (shown text of the underlying button matrix) of tab `id` during runtime the function `lv_tabview_set_tab_name(tabview, id, name)` can be used.

Tab button's position

By default, the tab selector buttons are placed on the top of the Tab view. It can be changed with `lv_tabview_set_btns_pos(tabview, LV_TABVIEW_TAB_POS_TOP/BOTTOM/LEFT/RIGHT/NONE)`

`LV_TABVIEW_TAB_POS_NONE` will hide the tabs.

Note that, you can't change the tab position from top or bottom to left or right when tabs are already added.

Animation time

The animation time is adjusted by `lv_tabview_set_anim_time(tabview, anim_time_ms)`. It is used when the new tab is loaded.

Scroll propagation

As the tabs' content object is a Page it can receive scroll propagation from an other Page-like object. For example, if a text area is created on the tab's content and that Text area is scrolled but it reached the end the scroll can be propagated to the content Page. It can be enabled with `lv_page/textarea_set_scroll_propagation(obj, true)`.

By default the tab's content Pages have enabled scroll propagation, therefore when they are scrolled horizontally the scroll is propagated to `LV_TABVIEW_PART_BG_SCRL` and this way the Pages will be scrolled.

The manual sliding can be disabled with `lv_page_set_scroll_propagation(tab_page, false)`.

Events

Besides the Generic events the following Special events are sent by the Slider:

- **LV_EVENT_VALUE_CHANGED** Sent when a new tab is selected by sliding or clicking the tab button

Learn more about *Events*.

Keys

The following *Keys* are processed by the Tabview:

- **LV_KEY_RIGHT/LEFT** Select a tab
- **LV_KEY_ENTER** Change to the selected tab

Learn more about *Keys*.

Example

API

Functions

lv_obj_t ***lv_tabview_create**(*lv_obj_t* *parent, lv_dir_t tab_pos, lv_coord_t tab_size)

lv_obj_t ***lv_tabview_add_tab**(*lv_obj_t* *tv, const char *name)

lv_obj_t ***lv_tabview_get_content**(*lv_obj_t* *tv)

lv_obj_t ***lv_tabview_get_tab_btns**(*lv_obj_t* *tv)

void **lv_tabview_set_act**(*lv_obj_t* *obj, uint32_t id, *lv_anim_enable_t* anim_en)

uint16_t **lv_tabview_get_tab_act**(*lv_obj_t* *tv)

Variables

const lv_obj_class_t **lv_tabview_class**

struct **lv_tabview_t**

Public Members

lv_obj_t **obj**

char ****map**

uint16_t **tab_cnt**

uint16_t **tab_cur**

lv_dir_t **tab_pos**

5.3.11 Tile view (lv_tileview)

Overview

The Tileview is a container object where its elements (called *tiles*) can be arranged in a grid form. By swiping the user can navigate between the tiles.

If the Tileview is screen sized it gives a user interface you might have seen on the smartwatches.

Parts and Styles

The Tileview has the same parts as Page. Expect `LV_PAGE_PART_SCRL` because it can't be referenced and it's always transparent. Refer the Page's documentation of details.

Usage

Valid positions

The tiles don't have to form a full grid where every element exists. There can be holes in the grid but it has to be continuous, i.e. there can't be an empty rows or columns.

With `lv_tileview_set_valid_positions(tileview, valid_pos_array, array_len)` the valid positions can be set. Scrolling will be possible only to this positions. The `0,0` index means the top left tile. E.g. `lv_point_t valid_pos_array[] = {{0,0}, {0,1}, {1,1}, {{LV_COORD_MIN, LV_COORD_MIN}}` gives a Tile view with "L" shape. It indicates that there is no tile in `{1,1}` therefore the user can't scroll there.

In other words, the `valid_pos_array` tells where the tiles are. It can be changed on the fly to disable some positions on specific tiles. For example, there can be a 2x2 grid where all tiles are added but the first row (y = 0) as a "main row" and the second row (y = 1) contains options for the tile above it. Let's say horizontal scrolling is possible only in the main row and not possible between the options in the second row. In this case the `valid_pos_array` needs to changed when a new main tile is selected:

- for the first main tile: `{0,0}, {0,1}, {1,0}` to disable the `{1,1}` option tile
- for the second main tile `{0,0}, {1,0}, {1,1}` to disable the `{0,1}` option tile

Set tile

To set the currently visible tile use `lv_tileview_set_tile_act(tileview, x_id, y_id, LV_ANIM_ON/OFF)`.

Add element

To add elements just create an object on the Tileview and position it manually to the desired position.

`lv_tileview_add_element(tielview, element)` should be used to make possible to scroll (drag) the Tileview by one its element. For example, if there is a button on a tile, the button needs to be explicitly added to the Tileview to enable the user to scroll the Tileview with the button too.

Scroll propagation

The scroll propagation feature of page-like objects (like List) can be used very well here. For example, there can be a full-sized List and when it reaches the top or bottom most position the user will scroll the tile view instead.

Animation time

The animation time of the Tileview can be adjusted with `lv_tileview_set_anim_time(tileview, anim_time)`.

Animations are applied when

- a new tile is selected with `lv_tileview_set_tile_act`
- the current tile is scrolled a little and then released (revert the original title)
- the current tile is scrolled more than half size and then released (move to the next tile)

Edge flash

An "edge flash" effect can be added when the tile view reached hits an invalid position or the end of tile view when scrolled.

Use `lv_tileview_set_edge_flash(tileview, true)` to enable this feature.

Events

Besides the Generic events the following Special events are sent by the Slider:

- **LV_EVENT_VALUE_CHANGED** Sent when a new tile loaded either with scrolling or `lv_tileview_set_act`. The event data is set ti the index of the new tile in `valid_pos_array` (It's type is `uint32_t *`)

Keys

- **LV_KEY_UP**, **LV_KEY_RIGHT** Increment the slider's value by 1
- **LV_KEY_DOWN**, **LV_KEY_LEFT** Decrement the slider's value by 1

Learn more about *Keys*.

Example

API

Functions

lv_obj_t ***lv_tileview_create**(*lv_obj_t* *parent)

Create a tileview objects

Parameters **par** -- pointer to an object, it will be the parent of the new tileview

Returns pointer to the created tileview

lv_obj_t ***lv_tileview_add_tile**(*lv_obj_t* *tv, uint8_t row_id, uint8_t col_id, lv_dir_t dir)

void **lv_obj_set_tile**(*lv_obj_t* *tv, *lv_obj_t* *tile_obj, *lv_anim_enable_t* anim_en)

void **lv_obj_set_tile_id**(*lv_obj_t* *tv, uint32_t col_id, uint32_t row_id, *lv_anim_enable_t* anim_en)

Variables

const lv_obj_class_t **lv_tileview_class**

const lv_obj_class_t **lv_tileview_tile_class**

struct **lv_tileview_t**

Public Members

lv_obj_t **obj**

struct **lv_tileview_tile_t**

Public Members

lv_obj_t **obj**

lv_dir_t **dir**

5.3.12 Window (lv_win)

Overview

The Window is container-like objects built from a header with title and button and a content area.

Parts and Styles

The main part is `LV_WIN_PART_BG` which holds the two other real parts:

1. `LV_WIN_PART_HEADER`: a header Container on the top with a title and control buttons
2. `LV_WIN_PART_CONTENT_SCRL` the scrollable part of a Page for the content below the header.

Besides these, `LV_WIN_PART_CONTENT_SCRL` has a scrollbar part called `LV_WIN_PART_CONTENT_SCRL`. Read the documentation of Page for more details on the scrollbars.

All parts supports the typical background properties. The title uses the *Text* properties of the header part.

The height of the control buttons is: *header height - header padding_top - header padding_bottom*.

Title

On the header, there is a title which can be modified by: `lv_win_set_title(win, "New title")`.

Control buttons

Control buttons can be added to the right of the window header with: `lv_win_add_btn_right(win, LV_SYMBOL_CLOSE)`, to add a button to the left side of the window header use `lv_win_add_btn_left(win, LV_SYMBOL_CLOSE)` instead. The second parameter is an Image source so it can be a symbol, a pointer to an `lv_img_dsc_t` variable or a path to file.

The width of the buttons can be set with `lv_win_set_btn_width(win, w)`. If `w == 0` the buttons will be square-shaped.

`lv_win_close_event_cb` can be used as an event callback to close the Window.

Scrollbars

The scrollbar behavior can be set by `lv_win_set_scrlbar_mode(win, LV_SCRLBAR_MODE_...)`. See Page for details.

Manual scroll and focus

To scroll the Window directly you can use `lv_win_scroll_hor(win, dist_px)` or `lv_win_scroll_ver(win, dist_px)`.

To make the Window show an object on it use `lv_win_focus(win, child, LV_ANIM_ON/OFF)`.

The time of scroll and focus animations can be adjusted with `lv_win_set_anim_time(win, anim_time_ms)`

Layout

To set a layout for the content use `lv_win_set_layout(win, LV_LAYOUT_...)`. See Container for details.

Events

Only the Generic events are sent by the object type.

Learn more about *Events*.

Keys

The following *Keys* are processed by the Page:

- **LV_KEY_RIGHT/LEFT/UP/DOWN** Scroll the page

Learn more about *Keys*.

Example

API

Functions

lv_obj_t ***lv_win_create**(*lv_obj_t* *parent, lv_coord_t header_height)

lv_obj_t ***lv_win_add_title**(*lv_obj_t* *win, const char *txt)

lv_obj_t ***lv_win_add_btn**(*lv_obj_t* *win, const void *icon, lv_coord_t btn_w)

lv_obj_t ***lv_win_get_header**(*lv_obj_t* *win)

lv_obj_t ***lv_win_get_content**(*lv_obj_t* *win)

Variables

const lv_obj_class_t **lv_win_class**

struct **lv_win_t**

Public Members

lv_obj_t **obj**

CHAPTER

SIX

LAYOUTS

6.1 Flex

6.2 Grid

CHAPTER

SEVEN

CONTRIBUTING

7.1 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 are new to the library or even new to programming.

It might be scary to make the first step but you have nothing to be afraid of. A friendly and helpful community is waiting for you. Get to know like-minded people and make something great together.

So let's find which contribution option fits you the best and help you join the development of LVGL!

Before getting started here are some guidelines to make contribution smoother:

- Be kind and friendly.
- Be sure to read the relevant part of the documentation before posting a question.
- Ask questions in the Forum and use GitHub for development-related discussions.
- Always fill out the post or issue templates in the Forum or GitHub (or at least provide equivalent information). It makes much easier to understand your case and you will get a useful answer faster.
- If possible send an absolute minimal but buildable code example in order to reproduce the issue. Be sure it contains all the required variable declarations, constants, and assets (images, fonts).
- Use Markdown to format your posts. You can learn it in 10 minutes.
- Speak about one thing in one issue or topic. It makes your post easier to find later for someone with the same question.
- Give feedback and close the issue or mark the topic as solved if your question is answered.
- For non-trivial fixes and features, it's better to open an issue first to discuss the details instead of sending a pull request directly.
- Please read and follow the Coding style guide.

7.2 Pull request

Merging new code into lvgl, documentation, blog, examples, and other repositories happen via *Pull requests* (PR for short). A PR is a notification like "Hey, I made some updates to your project. Here are the changes, you can add them if you want." To do this you need a copy (called fork) of the original project under your account, make some changes there, and notify the original repository about your updates. You can see how it looks like on GitHub for lvgl here: https://github.com/lvgl/lvgl/pulls.

To add your changes you can edit files online on GitHub and send a new Pull request from there (recommended for small changes) or add the updates in your favorite editor/IDE and use git to publish the changes (recommended for more complex updates).

7.2.1 From GitHub

1. Navigate to the file you want to edit.
2. Click the Edit button in the top right-hand corner.
3. Add your changes to the file
4. Add a commit message on the bottom of the page
5. Click the *Propose changes* button

7.2.2 From command line

The instructions describe the main `lvgl` repository but it works the same way for the other repositories.

1. Fork the lvgl repository. To do this click the "Fork" button in the top right corner. It will "copy" the `lvgl` repository to your GitHub account (`https://github.com/<YOUR_NAME>?tab=repositories`)
2. Clone your forked repository.
3. Add your changes. You can create a *feature branch* from *master* for the updates: `git checkout -b the-new-feature`
4. Commit and push 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`) 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.

7.3 Developer Certification of Origin (DCO)

7.3.1 Overview

To ensure that all licensing criteria is met for all repositories of the LVGL project we apply a process called DCO (Developer's Certificate of Origin).

The text of DCO can be read here: https://developercertificate.org/.

By contributing to any repositories of the LVGL project you state that your contribution corresponds with the DCO.

No further action is required if your contribution fulfills the DCO. If you are not sure about it feel free to ask us in a comment.

7.3.2 Accepted licenses and copyright notices

To make the DCO easier to digest, here are some practical guides about specific cases:

Your own work

The simplest case is when the contribution is solely your own work. In this case you can just send a Pull Request without worrying about any licensing issues.

Use code from online source

If the code you would like to add is based on an article, post or comment on a website (e.g. StackOverflow) the license and/or rules of that site should be followed.

For example in case of StackOwerflow a notice like this can be used:

```
/* The original version of this code-snippet was published on StackOverflow.
 * Post: http://stackoverflow.com/questions/12345
 * Author: http://stackoverflow.com/users/12345/username
 * The following parts of the snippet were changed:
 * - Check this or that
 * - Optimize performance here and there
 */
 ... code snippet here ...
```

Use MIT licensed code

As LVGL is also MIT licensed other MIT licensed code can be integrated without issues. The MIT license requests a copyright notice be added to the derived work. So you need to copy the original work's license file or it's text to the code you want to add.

Use GPL licensed code

As GPL license is not compatible with MIT license so LVGL can not accept GPL licensed code.

7.4 When you get started with LVGL

Even if you're just getting started with LVGL there are plenty of ways to get your feet wet. Most of these options don't even require knowing a single line of code of LVGL.

7.4.1 Give LVGL a Star

Show that you like LVGL by giving it star on GitHub!

Star

This simple click makes LVGL more visible on GitHub and makes it more attractive to other people. So with this, you already helped a lot!

7.4.2 Tell what you have achieved

Have you already started using LVGL in a *Simulator*, a development board, or your custom hardware? Was it easy or were there some obstacles? Are you happy with the result?

If so why don't you tell it to your friends? You can post it on Twitter, Facebook, LinkedIn, or create a YouTube video.

Any of these helps a lot to spread the word of LVGL and familiarize it with new developers.

Only thing: don't forget to add a link to `https://lvgl.io` or `https://github.com/lvgl` and `#lvgl`. Thank you! :)

7.4.3 Write examples

As you learn LVGL probably you will play with the features of widgets. But why don't you publish your experiments?

Every widgets' documentation contains some examples. For example here are the examples of the *Drop-down list*. The examples are directly loaded from the lv_examples repository.

So all you need to do is send a *Pull request* to the lv_examples repository and follow some conventions:

- Name the examples like `lv_ex_<widget_name>_<id>`
- Make the example as short and simple as possible
- Add comments to explain what the example does
- Use 320x240 resolution
- Create a screenshot about the example
- Update `index.rst` in the example's folder with your new example. See how the other examples are added.

7.4.4 Improve the docs

As you read the documentation you might see some typos or unclear sentences. For typos and straightforward fixes, you can simply edit the file on GitHub. There is an `Edit on Github` link on the top right-hand corner of all pages. Click it to see the file on GitHub, hit the Edit button, and add your fixes as described in *Pull request - From GitHub* section.

Note that the documentation is also formatted in Markdown.

7.4.5 Translate the docs

If you have more free time you can even translate the documentation. The currently available languages are shown in the locals folder.

If your chosen language is still not added, please write a comment here.

To add your translations:

- Find the *.po* in `<language_code>/LC_MESSAGES/<section_name>.po`. E.g. the widgets translated to German should be in `de/LC_MESSAGES/widgets.po`.
- Open a po file and fill the `msgstr` fields with the translation
- Send a *Pull request*

To display a translation in the public documentation page at least these sections should be translated:

- Get started: Quick overview
- Overview: Objects, Events, Styles
- Porting: System overview, Set-up a project, Display interface, Input device Interface, Tick interface
- 5 widgets of your choice

7.4.6 Write a blog post

The LVGL Blog welcomes posts from anyone. It's a good place to talk about a project you created with LVGL, write a tutorial, or share some nice tricks. The latest blog posts are shown on the homepage of LVGL to make your work more visible.

The blog is hosted on GitHub. If you add a post GitHub automatically turns it into a website. See the README of the blog repo to see how to add your post.

7.5 When you already use LVGL

7.5.1 Give feedback

Let us know what you are working on! You can open a new topic in the My projects category of the Forum. Showing your project to others is a win-win situation because it increases your and LVGL's reputation at the same time.

If you don't want to speak about it publicly feel free to use Contact form on lvgl.io to private message to us.

7.5.2 Report bugs

As you use LVGL you might find bugs. Before reporting them be sure to check the relevant parts of the documentation.

If it really seems like a bug feel free to open an issue on GitHub.

When filing the issue be sure to fill the template. It helps a lot to find the root of the problems and helps to avoid a lot of questions.

7.5.3 Send fixes

The beauty of open-source software is you can easily dig in to it to understand how it works. You can also fix or adjust it as you wish.

If you found and fixed a bug don't hesitate to send a *Pull request* with the fix.

In your Pull request please also add a line to `CHANGELOG.md`.

7.5.4 Join the conversations in the Forum

It feels great to know you are not alone if something is not working. It's even better to help others when they struggle with something.

While you were learning LVGL you might have questions and used the Forum to get answers. As a result, you probably have more knowledge about how LVGL works.

One of the best ways to give back is to use the Forum and answer the questions of newcomers - like you were once.

Just read the titles and if you are familiar with the topic don't hesitate to share your thoughts and suggestions.

Participating in the discussions is one of the best ways to become part of the project and get to know like-minded people!

7.5.5 Add features

We collect the planned features in GitHub issues tracker and mark them with Help wanted label. If you are interested in any of them feel free to share your opinion and/or participate in the the implementation.

Other features which are (still) not on the road map are listed in the Feature request category of the Forum. If you have a feature idea for LVGL please use the Forum to share it! Make sure to check that there isn't an existing post; if there is, you should comment on it instead to show that there is increased interest in an existing request.

When adding a new features the followings also needs to be updated:

- Add a line to CHANGELOG.md.
- Update the documentation. See this *guide*.
- Add an example if applicable. See this *guide*.

7.6 When you are confident with LVGL

7.6.1 Become a maintainer

If you want to become part of the core development team, you can become a maintainer of a repository.

By becoming a maintainer:

- you get write access to that repo:
  - add code directly without sending a pull request
  - accept pull requests
  - close/reopen/edit issues
- your name will be added in the credits section of lvgl.io/about (will be added soon) and lvgl's README.
- you can join the Core_contributor group in the Forum and get the LVGL logo on your avatar.
- your word has higher impact when we make decisions

You can become a maintainer by invitation, however the following conditions need to met

1. Have > 50 replies in the Forum. You can look at your stats here
2. Send > 5 non-trivial pull requests to the repo where you would like to be a maintainer

If you are interested, just send a message (e.g. from the Forum) to the current maintainers of the repository. They will check is 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.

7.6.2 Move your project repository under LVGL organization

Besides the core `lvgl` repository there are other repos for ports to development boards, IDEs or other environment. If you ported LVGL to a new platform we can host it under the LVGL organization among the other repos.

This way your project will become part of the whole LVGL project and can get more visibility. If you are interested in this opportunity just open an issue in lvgl repo and tell what you have!

If we agree that your port is useful, we will open a repository for your project where you will have admin rights.

To make this concept sustainable there a few rules to follow:

- You need to add a README to your repo.
- We expect to maintain the repo to some extent:
  - Follow at least the major versions of lvgl
  - Respond to the issues (in a reasonable time)
- If there is no activity in a repo for 6 months it will be archived

CHAPTER

EIGHT

CHANGELOG

8.1 v7.11.0

8.1.1 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)

8.1.2 Bugfixes

- fix(gauge) fix needle invalidation
- fix(bar) correct symmetric handling for vertical sliders

8.2 v7.10.1 (Planned for 16.02.2021)

8.2.1 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

8.3 v7.10.0

8.3.1 New features

- feat(indev) allow input events to be passed to disabled objects
- feat(spinbox) add inline get_step function for MicroPython support

8.3.2 Bugfixes

- fix(btnmatrix) fix lv_btnmatrix_get_active_btn_text() when used in a group

8.4 v7.9.1

8.4.1 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

8.5 v7.9.0

8.5.1 New features

- feat(chart) add lv_chart_remove_series and lv_chart_hide_series
- feat(img_cahce) allow disabling image caching
- calendar: make get_day_of_week() public
- Added support for Zephyr integration

8.5.2 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(linemeter) 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

8.6 v7.8.1

8.6.1 Bugfixes

- fix(lv_scr_load_anim) fix when multiple screen are loaded at tsame time with delay
- fix(page) fix LV_SCOLLBAR_MODE_DRAG

8.7 v7.8.0 (01.12.2020)

8.7.1 New features

- make DMA2D non blocking
- add unscii-16 built-in font
- add KConfig
- add lv_refr_get_fps_avg()

8.7.2 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(linemeter) 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

8.8 v7.7.2 (17.11.2020)

8.8.1 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 a the closest available state with valid src
- fix(spinbox) update cursor position in lv_spinbox_set_step

8.9 v7.7.1 (03.11.2020)

8.9.1 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 obejcts correctly

8.10 v7.7.0 (20.10.2020)

8.10.1 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

8.10.2 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()`

8.11 v7.6.1 (06.10.2020)

8.11.1 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

8.12 v7.6.0 (22.09.2020)

8.12.1 New features

- Check whether any style property has changed on a state change to decide if any redraw is required

8.12.2 Bugfixes

- Fix selection of options with non-ASCII letters in dropdown list
- Fix font loader to support LV_FONT_FMT_TXT_LARGE

8.13 v7.5.0 (15.09.2020)

8.13.1 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

8.13.2 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 `decmopr_buf` to GC roots
- Fix divisioin 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

8.14 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.

8.14.1 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.

8.14.2 Bugfixes

- Fix color bleeding on border drawing
- Fix using 'LV_SCROLLBAR_UNHIDE' after 'LV_SCROLLBAR_ON'
- Fix croping of last column/row if an image is zoomed
- Fix zooming and rotateing 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

8.15 v7.3.1 (18.08.2020)

8.15.1 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 (udnerline 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

8.16 v7.3.0 (04.08.2020)

8.16.1 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`

8.16.2 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: hadle combined states of buttons (e.g. chacked + disabled)
- textarea: fix typo in lv_textarea_set_sscrollbar_mode
- gauge: fix image needle drawing
- fix using freed memory in _lv_style_list_remove_style

8.17 v7.2.0 (21.07.2020)

8.17.1 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

8.17.2 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

8.18 v7.1.0 (07.07.2020)

8.18.1 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.

8.18.2 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.

8.19 v7.0.2 (16.06.2020)

8.19.1 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 the 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 `scrllable` -> `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

8.20 v7.0.1 (01.06.2020)

8.20.1 Bugfixes

- Make the Microptyhon 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

8.21 v7.0.0 (18.05.2020)

8.21.1 Documentation

The docs for v7 is available at https://docs.littlevgl.com/v7/en/html/index.html

8.21.2 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.

8.21.3 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.

8.21.4 New style system

The old style system is replaced with a new more flexible and lightweighted 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 and 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 lighweighted text on buttons too.
- *margin*: similar to *padding* but used to keep space outside of the object

Read the Style section of the documentation to learn how the new styles system works.

8.21.5 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.

8.21.6 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

8.21.7 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 bacground 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

8.21.8 New object types

- `lv_objmask`: masks can be added to it. The children will be masked accordingly.

8.21.9 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`

8.21.10 Demos

- lv_examples was reworked and new examples and demos were added

8.21.11 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.

8.21.12 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.

CHAPTER

NINE

ROADMAP

This is a summary for thenew fatures of the major releases and a collection of ideas.

This list indicates only the current intention and can be changed.

9.1 v8

Planned to May 2021

- Create an `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
- New scrolling:
  - See feat/new-scroll branch and #1614) issue.
  - Remove `lv_page` and support scrolling on `lv_obj`
  - Support "elastic" scrolling when scrolled in
  - Support scroll chaining among any objects types (not only `lv_pages`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
  - Already working
- New layouts:
  - See #1615 issue
  - CSS Grid-like layout support
  - CSS Flexbox-like layout support
  - Remove `lv_cont` and support layouts on `lv_obj`
- Simplified File system interface (feat/new_fs_api branch) to make porting easier
  - Work in progress
- Remove the align parameter from `lv_canvas_draw_text`
- Remove the copy parameter from create functions

- Optimize and simplifie styles #1832
- Use a more generic inheritenace #1919

9.2 v8.x

- Add radio button widget
- Unit testing (gtest?). See #1658
- Benchmarking (gem5?). See #1660
- chart: pre-delete X pint after the lastly set
- chart: autoscroll to the right
- `lv_snapshot`: buffer a widget and all of its children into an image. he source widget can be on a different screen too. The result image can be transformed.
- 9-patch support for `lv_imgbtn`.
- Handle stride. See #1858
- Rework animation to something like GSAP
  - Add scroll trigger inspired by GSAP scrolltrigger
  - Add FLIP inspired by GSAP FLIP

9.3 v9

- Simplify `groups`. Discussion is here.
- Consider direct binary font format support
- Optimize line and circle drawing and masking
- Reconsider color format management for run time color format setting, and custom color format usage. (Also RGB888)
- Switch to RGBA colors in styles
- Make gradients more versatile
- Make image transformations more versatile

9.4 Ideas

- Use generate-changelog to automatically generate changelog
- lv_mem_alloc_aligned(size, align)
- Text node. See #1701
- CPP binding. See Forum
- Optimize font decompression
- Need coverage report for tests

- Need static analyze (via coverity.io or somehing else)
- Support dot_begin and dot_middle long modes for labels
- Add new label alignment modes. #1656
- Support larger images: #1892

INDEX

Symbols

L