LittlevGL documentation (Turkish)

Table of contents

Hoşgeldiniz

Written for v5.3, revision 2

Welcome LittlevGL Embedded GUI Library

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

Key features

- **Powerful building blocks** buttons, charts, lists, sliders, images, etc.

- **Advanced graphics** with animations, anti-aliasing, opacity, smooth scrolling

- **Various input devices** touch pad, mouse, keyboard, encoder, buttons, etc.

- **Multi-language support** with UTF-8 encoding

- **Fully customizable** graphical elements

- **Hardware independent** to use with any microcontroller or display

- **Scalable** to operate with little memory (50 kB Flash, 10 kB RAM)

- **OS, External memory and GPU** supported but not required

- **Single frame buffer** operation even with advances graphical effects

- **Written in C** for maximal compatibility

- **Simulator** to develop on PC without embedded hardware

- **Tutorials, examples, themes** for rapid development

How to get started?

Read the documentation

Reading the documentation first is always a good idea. Don't be afraid it takes only a little time to learn the basics.

You can start here with Porting or with an Introduction to the library.

Tutorial

There is a tutorial to show you the most important parts step-by-step.

It's located in the lv_examples repository

Try LittlevGL in a PC simulator

If you don't have an embedded hardware with a display yet you can try the library on PC. The PC simulator uses a window on your monitor to simulate a display and uses your mouse to click on that display.

The simulator is works with Windows, Linux, and OSX as well.

Here you can learn how to set-up the simulator: PC simulator

Contribute

Use GitHub's issue tracker to:

- Report bugs

- Suggest new features

- Add new features

- Help others

Before contributing, read the related document: CONTRIBUTING

Porting

Written for v5.2

System architecture

Application

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

LittlevGL

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

Driver

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

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

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).

External display controller

If the MCU doesn't have TFT/LCD driver 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.

Requirements

- **16, 32 or 64 bit** microcontroller or processor

- **16 MHz** clock speed

- **8 kB RAM** for static data and **>2 KB RAM** for dynamic data (graphical objects)

- **64 kB program memory** (flash)

- **Optionally ~1/10 screen sized memory** for internal buffering (at 240 × 320, 16 bit colors it means 15 kB)

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

- No external RAM required (but supported)

- No float numbers are used

- No GPU needed (but supported)

- Only a single frame buffer is required located in:
  - Internal RAM or

- External RAM or

- External display controller's memory

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

- Disable the unused object types to save RAM and ROM

- Change the size of the graphical buffer to save RAM

- Use simpler styles to reduce the rendering time

Project set-up

Get the library

The Littlev Graphics Library is available on GitHub: https://github.com/littlevgl/lvgl. You can clone or download the latest version of the library from here or you can use the Download page as well.

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

Config file

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

Copy _lvgl/lv_conf_templ.h_ next to the *lvgl* directory and rename it to _lv_conf.h_. Open the file and delete the first *#if* and the last *#endif* to enable the content. In the config file comments explain the meaning of the options. Check at least these three config options and modify them according to your hardware:

1. **LV_HOR_RES** Your display's horizontal resolution

2. **LV_VER_RES** Your display's vertical resolution

3. **LV_COLOR_PETH** 8 for (RG332), 16 for (RGB565) or 32 for (RGB888 and ARGB8888).

Initialization

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

1. Call _lv_init()_

2. Initialize your drivers

3. Register the display and input devices drivers in LittlevGL. (see below)

Porting the library

To adopt LittlevGL into your project firstly you have to provide some functions and register them in the graphics library.

Display interface

To set up a display an **lv_disp_drv_t** variable has to be initialized:

```
lv_disp_drv_t disp_drv;
lv_disp_drv_init(&disp_drv); /*Basic initialization*/
disp_drv. ...=...; /*Initialize the fields here. See below.*/
disp_drv_register(&disp_drv); /*Register the driver in LittlevGL*/
```

You can configure the driver for different operation modes. To learn more about the drawing modes visit Drawing and rendering.

Internal buffering (VDB)

The graphics library works with an internal buffering mechanism to create advances graphics features with only one frame buffer. The internal buffer is called VDB (Virtual Display Buffer) and its size can be adjusted in lv_conf.h with _LV_VDB_SIZE_. When _LV_VDB_SIZE_ > 0 then the

internal buffering is used and you have to provide a function which flushes the buffer's content to your display:

```
disp_drv.disp_flush = my_disp_flush;
...
void my_disp_flush(int32_t x1, int32_t y1, int32_t x2, int32_t y2, const lv_color_t* color_p)
{
    /*TODO Copy 'color_p' to the specified area*/
    /*Call 'lv_fluh_ready()' when ready*/lv_flush_ready();
}
```

In the flush function, you can use DMA or any hardware to do the flushing in the background, but when the flushing is ready you have to call

```
lv_flush_ready();
```

Hardware acceleration (GPU)

First of all using GPU is totally optional. But if your MCU supports graphical acceleration then you can use it. The _mem_blend_ and _mem_fill_ fields of the display driver is used to interface with a GPU. The GPU related functions can be used only if internal buffering (VDB) is enabled.

```
disp_drv.mem_blend = my_mem_blend;     /*Blends two color arrays using opacity*/
disp_drv.mem_fill = my_mem_fill;       /*Fills an array with a color*/
...
void my_mem_blend(lv_color_t* dest, constlv_color_t* src, uint32_t length, lv_opa_t opa)
{
  /*TODO Copy 'src' to 'dest' but blend it with 'opa' alpha */
}

void my_mem_fill(lv_color_t* dest, uint32_t length, lv_color_t color)
{
  /*TODO Fill 'length' pixels in 'dest' with 'color'*/
}
```

Unbuffered drawing

It is possible to draw directly to a frame buffer when the internal buffering is disabled (LV_VDB_SIZE = 0).

```
disp_drv.disp_fill = my_disp_fill;/*Fill an area in the frame buffer*/
disp_drv.disp_map = my_disp_map;/*Copy a color_map (e.g. image) into the frame buffer*/
...
void my_disp_map(int32_t x1,int32_t y1,int32_t x2,int32_t y2,constlv_color_t* color_p)
{
  /*TODO Copy 'color_p' to the specified area*/
}

voidmy_disp_fill(int32_t x1,int32_t y1,int32_t x2,int32_t y2,lv_color_t color)
{
  /*TODO Fill the specified area with 'color'*/
}
```

Keep in mind this way during refresh some artifacts can be visible because the layers are drawn after each other. And some high-level graphics features like anti-aliasing, opacity or shadows aren't available in this configuration.

If you use an external display controller which supports accelerated filling (e.g. RA8876) then you can use this feature in _disp_fill()_

Input device interface

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

```
lv_indev_drv_t indev_drv;lv_indev_drv_init(&indev_drv);  /*Basic initialization*/
indev_drv.type =.../*See below.*/
indev_drv.read =.../*See below.*/
lv_indev_drv_register(&indev_drv);  /*Register the driver in LittlevGL*/
```

type can be

- LV_INDEV_TYPE_POINTER: touchpad or mouse

- LV_INDEV_TYPE_KEYPAD: keyboard

- LV_INDEV_TYPE_ENCODER: left, right, push

- LV_INDEV_TYPE_BUTTON: external buttons pressing the screen

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

To learn more about input devices visit Input devices.

Touchpad, mouse or any pointer

```
indev_drv.type = LV_INDEV_TYPE_POINTER;
indev_drv.read = my_input_read;
```

The read function should look like this:

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

> **IMPORTANT NOTE:** Touchpad drivers must return the last X/Y coordinates even when the state is `LV_INDEV_STATE_REL` .

Keypad or keyboard

```
indev_drv.type = LV_INDEV_TYPE_KEYPAD;
indev_drv.read = my_input_read;
```

The read function should look like this:

```
bool keyboard_read(lv_indev_data_t*data){
  data->key = last_key();          /*Set the last pressed or released key*/
  if(key_pressed()){
    data->state = LV_INDEV_STATE_PR;
  }
  else{
    data->state = LV_INDEV_STATE_REL;
  }
  return false; /*No buffering so no more data read*/
}
```

To use a keyboard:

- Register a *read* function (like above) with _LV_INDEV_TYPE_KEYPAD_ type.

- _USE_LV_GROUP_ has to be enabled in _lv_conf.h_

- An object group has to be created: _lv_group_create()_ and objects have to be added: _lv_group_add_obj()_

- The created group has to be assigned to an input device: _lv_indev_set_group(my_indev, group1);_

- Use _LV_GROUP_KEY_..._ to navigate among the objects in the group

Visit Touchpad-less navigation to learn more.

Encoder

With an encoder you can do 4 things:

1. press its button

2. long press its button

3. turn left

4. turn right

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

navigate inside the object. To leave edit mode press long the button.

```
indev_drv.type = LV_INDEV_TYPE_ENCODER;
indev_drv.read = my_input_read;
```

The read function should look like this:

```
bool encoder_read(lv_indev_data_t*data){
  data->enc_diff = enc_get_new_moves();
  if(enc_pressed()){
    data->state = LV_INDEV_STATE_PR;
  }
  else{
    data->state = LV_INDEV_STATE_REL;
  }

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

- To use an `ENCODER`, similarly to the `KEYPAD`, the objects should be added to groups

Button

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

The read function should look like this:

```
bool button_read(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?*/
      last_btn = btn_pr;            /*Save the ID of the pressed button*/
      data->state = LV_INDEV_STATE_PR;  /*Set the pressed state*/
    } else {
      data->state = LV_INDEV_STATE_REL; /*Set the released state*/
    }

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

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

- The buttons need to be assigned to pixels on the screen using `lv_indev_set_button_points(indev, points_array)`. Where _points_array_ look like `const lv_point_t points_array[] = { {12,30},{60,90}, ...}`

Tick interface

The LittlevGL uses a system tick. Call the `lv_tick_inc(tick_period)` function periodically and tell the call period in milliseconds. For example if called in every milliseconds: `lv_tick_inc(1)`.

Task handling

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

- *while(1)* of *main()* function

- timer interrupt periodically

- 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);
}
```

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 in 1 sec*/
  if(lv_indev_get_inactive_time(NULL) < 1000) {
    lv_task_handler();
  }
  /*Sleep after 1 sec inactivity*/
  else {
    timer_stop();              /*Stop the timer where lv_tick_inc() is called*/
    sleep();           /*Sleep the MCU*/
  }
  my_delay_ms(5);
}
```

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

```
/*Force task execution on wake-up*/
lv_tick_inc(LV_REFR_PERIOD);
timer_start();                 /*Restart the timer where lv_tick_inc() is called*/
lv_task_handler();
```

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

Using with an operating system

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

The **simple scenario** is to don't use the operating system's tasks but use `lv_task` s. An _lv_task_ is a function called periodically in `lv_task_handler`. In the _lv_task_ you can get the state of the sensors, buffers etc and call LittlevGL functions to refresh the GUI. To create an _lv_task_ use: `lv_task_create(my_func, period_ms, LV_TASK_PRIO_LOWEST/LOW/MID/HIGH/HIGHEST, custom_ptr)`

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

Porting example

Here you will find an example porting code: Porting tutorial.

PC Simulator

Written for v5.1

You can try out the Littlev Graphics Library **using only your PC** without any development board. Write a code, run it on the PC and see the result on the monitor. It is cross-platform: Windows, Linux and OSX are supported!

- Needs only few minutes setup

- Costs $0. No PCB cost and no pay for any software

- A TFT display is simulated and shown on the monitor of your PC

- The touch pad is replaced by your mouse

- The written code is portable, you can simply copy it when using an embedded hardware

Install Eclipse CDT

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

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

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

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

Install SDL 2

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

Linux

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

1. Find the current version of SDL2: `apt-cache search libsdl2 (e.g. libsdl2-2.0-0)`

2. Install SDL2: `sudo apt-get install libsdl2-2.0-0` (replace with the found version)

3. Install SDL2 development package: `sudo apt-get install libsdl2-dev`

4. If build essentials are not installed yet: `sudo apt-get install build-essential`

Windows

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

1. Download the development libraries of SDL.
   Go to https://www.libsdl.org/download-2.0.php and download *Development Libraries: SDL2-devel-2.0.5-mingw.tar.gz*

2. Uncompress the file and go to _x86_64-w64-mingw32_ directory (for 64 bit MinGW) or to *i686-w64-mingw32* (for 32 bit MinGW)

3. Copy _...*mingw32/include/SDL2* folder to _C:/MinGW/.../x86_64-w64-mingw32/include_

4. Copy _...*mingw32/lib/* content to _C:/MinGW/.../x86_64-w64-mingw32/lib_

5. Copy _...*mingw32/bin/SDL2.dll* to _{eclipse_worksapce}/pc_simulator/Debug/_. Do it later when Eclipse is installed.

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

OSX

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

If something is not working I suggest this tutorial to get started with SDL.

Pre-configured project

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

Add the pre-configured project to Eclipse CDT

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

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

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

- Copy the **SDL2.dll** into the project's Debug folder

- Righ click on the project -> Project properties -> C/C++ Build -> Settings -> Libraries -> Add ... and add *mingw32* above SDLmain and SDL. (The order is important: mingw32, SDLmain, SDL)

Compile and Run

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

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

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

Next step

To create your first LittlevGL GUI you should read the pages below Porting in the sidebar.

Giriş Aygıtları

Written for v5.1

To interact with the created object *Input devices* are required. For example Touchpad, Mouse, Keyboard or even an Encoder. To learn how to add an input device, read the Porting guide.

When you register an input device driver the library adds some extra information to it to describe the state of the input device in more detail. When a user action (e.g. a button press) happens and an action (callback) function is triggered always there is an input device which triggered that action. You can get this input device with

```
lv_indev_t *indev = lv_indev_get_act();
```

It might be important when you need to know some special information about the input device like the currently pressed point, or dragging an object or not etc.

The input devices have a very simple API:

```
/*Get the last point on a display input*/
void lv_indev_get_point(lv_indev_t * indev, point_t * point);

/*Check if there is dragging on input device or not */
bool lv_indev_is_dragging(lv_indev_t * indev);

/*Get the vector of dragging on a input device*/
void lv_indev_get_vect(lv_indev_t * indev, point_t * point);

/*Do nothing until the next release*/
void lv_indev_wait_release(lv_indev_t * indev);

/*Do nothing until the next release*/
void lv_indev_wait_release(lv_indev_t * indev);

/*Reset one or all (use NULL) input devices*/
void lv_indev_reset(lv_indev_t * indev);

/*Reset the long pressed state of an input device*/
void lv_indev_reset_lpr(lv_indev_t * indev);

/*Set a cursor for a pointer input device*/
void lv_indev_set_cursor(lv_indev_t * indev, lv_obj_t * cur_obj);

/*Set a destination group for a keypad input device*/
void lv_indev_set_group(lv_indev_t * indev, lv_group_t * group);
```

Touchpad-less navigation

The objects can be grouped in order to easily **control** them **without touchpad or mouse**. It allows you to use

- Keyboard or keypad

- Hardware buttons

- Encoder

to navigate among objects.

Firstly you have to **create an object group** with `lv_group_t *group = lv_group_create()` and add objects to it with `lv_group_add_obj(group, obj)`. In a group always there is a *focused* object. All the button press will be "sent" to the currently focused object.

To navigate among the objects in a group (change the focused object) and interact with them an _LV_INDEV_TYPE_KEYPAD_ typed input device is required. In its *read* function you can tell the library which key is pressed or released. To learn how to add an input device, read the Porting guide.

Besides you have to **assign the group to the input device** with

```
lv_indev_set_group(indev, group)
```

There are some special **control characters** which can be used in the *read* function:

- **LV_GROUP_KEY_NEXT** Focus on the next object

- **LV_GROUP_KEY_PREV** Focus on the previous object

- **LV_GROUP_KEY_UP** Increment the value, move up or click the focused object (move up means e.g. select an upper list element)

- **LV_GROUP_KEY_DOWN** Decrement the value or move down on the focused object (move down means e.g. select a lower list element)

- **LV_GROUP_KEY_RIGHT** Increment the value or click the focused object

- **LV_GROUP_KEY_LEFT** Decrement the value of the focused object

- **LV_GROUP_KEY_ENTER** Click the focused object or a selected element (e.g. list element)

- **LV_GROUP_KEY_ESC** Close the object (e.g. drop-down list)

In some cases (e.g. when a pop-up window appears) it is useful to freeze the focus on an object. It means the _LV_GROUP_KEY_NEXT/PREV_ will be ignored. You can do it with `lv_group_focus_freeze(group,true)`.

The **style of the object in focus** is modified by a function. By default, it makes the object's colors orangish but you can also specify your own style updater function in each group with

```
void lv_group_set_style_mod_cb(group, style_mod_cb).
```

The _style_mod_cb_ needs an `lv_style_t *` parameter which is a copy of the focused object's style. In the callback, you can mix some colors to the current ones, and modify parameters but **it is not permitted to set attributes which modify the size** (like _letter_space_, *padding* etc.)

Renkler

Written for v5.1

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 (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_color24_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 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_color24_t_ types have got four fields:

- **red** red channel

- **green** green channel

- **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 24.

You can **convert a color from the current color depth** to an other. 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_color24_t c24;
c24.full = lv_color_to24(c);     /*Give a 32 bit number with the converted color*/
```

You can **create a color** 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(0xFF8080)` or `my_color = LV_COLOR_HEX3(0xF88)`.

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.

To describe **opacity** the _lv_opa_t_ type is created as wrapper to _uint8_t_. Some defines are also introduced:

- **LV_OPA_TRANSP** Value: 0, means the opacity makes the color fully 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 fully covers

- **LV_OPA_COVER** Value: 255, means the color fully covers

You can also use the _LV_OPA_*_ defines in _lv_color_mix() as ratio._

The color module defines the **most basic colors**:

- `LV_COLOR_BLACK`
- `LV_COLOR_GRAY`
- `LV_COLOR_SILVER`
- `LV_COLOR_RED`
- `LV_COLOR_MARRON`
- `LV_COLOR_LIME`
- `LV_COLOR_GREEN`
- `LV_COLOR_OLIVE`
- `LV_COLOR_BLUE`
- `LV_COLOR_NAVY`
- `LV_COLOR_TAIL`
- `LV_COLOR_CYAN`
- `LV_COLOR_AQUA`
- `LV_COLOR_PURPLE`
- `LV_COLOR_MAGENTA`
- `LV_COLOR_ORANGE`
- `LV_COLOR_YELLOW`

as well as `LV_COLOR_WHITE` .

Fontlar

Written for v5.1

In LittlevGL fonts are bitmaps and other descriptors to store the images of the letters (glyph) and some additional information. A font is stored in a **lv_font_t** variable and can be set it in style's *text.font* field.

The fonts have a **bpp (Bit-Per-Pixel)** property. It shows how much bit is used to describe a pixel in the font. The value stored for a pixel determines the pixel's opacity. This way the image of the letters (especially on the edges) can be smooth and even. The possible bpp values are 1, 2, 4 and 8 (higher value means better quality). The bpp also affects the required memory size to store the font. E.g. bpp = 4 makes the font's memory size 4 times greater compared to bpp = 1.

Built-in fonts

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

- 10 px
- 20 px
- 30 px
- 40 px

You can enable the fonts with 1, 2, 4 or 8 values to set its bpp (e.g. `#define USE_LV_FONT_DEJAVU_20 4` in `lv_conf.h`).

The built-in fonts exist with **multiply character-sets** in each size:

- ASCII (Unicode 32..126)
- Latin supplement (Unicode 160..255)
- Cyrillic (Unicode 1024..1279)

The built-in fonts use the *Dejavu* font.

The built-in fonts are **global variables** with names like:

- `lv_font_dejavu_20` (20 px ASCII font)
- `lv_font_dejavu_20_latin_sup` (20 px Latin supplement font)
- `lv_font_dejavu_20_cyrillic` (20 px Cyrillic font)

Unicode support

The LittlevGL supports Unicode letter from **UTF-8** coded characters. You need to configure your editor to save your code/text as UTF-8 (usually this the default) and enable _LV_TXT_UTF8_ in lv_conf.h. Without enabled _LV_TXT_UTF8_ only ASCII fonts and symbols can be used (see the symbols below)

After it the texts will be decoded to determine the Unicode values. To display the letters your font needs to contain the image (glyph) of the characters.

You can assign more fonts to create a **larger character-set**. To do this choose a base font (typically the ASCII font) and add the extensions to it: `lv_font_add(child, parent)`. Only fonts with the same height can be assigned.

The built-in fonts are already added to the same sized ASCII font. For example if _USE_LV_FONT_DEJAVU_20_ and _USE_LV_FONT_DEJAVU_20_LATIN_SUP_ are enabled in `lv_conf.h` then the *"abcÁÖÜ"* text can be rendered when using _lv_font_dejavu_20_.

Symbol fonts

The symbol fonts are special fonts which contain symbols instead of letters. There are **built-in symbol fonts** as well and they are also assigned to the ASCII font with the same size. In a text, a symbol can be referenced like _SYMBOL_LEFT_, _SYMBOL_RIGHT_ etc. You can mix these symbol names with strings:

```
lv_label_set_text(label1,"Right "SYMBOL_RIGHT);
```

The symbols can be used without UTF-8 support as well. (_LV_TXT_UTF8   0_)

The list above shows the existing symbols:

♫	SYMBOL_AUDIO	✚	SYMBOL_PLUS
▦	SYMBOL_VIDEO	━	SYMBOL_MINUS
☰	SYMBOL_LIST	⚠	SYMBOL_WARNING
✔	SYMBOL_OK	⤬	SYMBOL_SHUFFLE
✖	SYMBOL_CLOSE	∧	SYMBOL_UP
⏻	SYMBOL_POWER	∨	SYMBOL_DOWN
⚙	SYMBOL_SETTINGS	⟲	SYMBOL_LOOP
⏏	SYMBOL_TRASH	▮	SYMBOL_DIRECTORY
⌂	SYMBOL_HOME	⬆	SYMBOL_UPLOAD
⤓	SYMBOL_DOWNLOAD	☎	SYMBOL_CALL
⬒	SYMBOL_DRIVE	✂	SYMBOL_CUT
⟳	SYMBOL_REFRESH	⎘	SYMBOL_COPY
◀	SYMBOL_MUTE	🖫	SYMBOL_SAVE
🔉	SYMBOL_VOLUME_MID	⚡	SYMBOL_CHARGE
🔊	SYMBOL_VOLUME_MAX	🔔	SYMBOL_BELL
🖼	SYMBOL_IMAGE	⌨	SYMBOL_KEYBOARD
✎	SYMBOL_EDIT	➤	SYMBOL_GPS
⏮	SYMBOL_PREV	🗋	SYMBOL_FILE
▶	SYMBOL_PLAY	📶	SYMBOL_WIFI
⏸	SYMBOL_PAUSE	🔋	SYMBOL_BATTERY_FULL
■	SYMBOL_STOP	🔋	SYMBOL_BATTERY_3
⏭	SYMBOL_NEXT	🔋	SYMBOL_BATTERY_2
⏏	SYMBOL_EJECT	🔋	SYMBOL_BATTERY_1
‹	SYMBOL_LEFT	🔋	SYMBOL_BATTERY_EMPTY
›	SYMBOL_RIGHT	❈	SYMBOL_BLUETOOTH

Add new font

If you want to **add new fonts to the library** you can use the Online Font Converter Tool. It can create a C array from a TTF file which can be copied copy to your project. You can specify the height, the range of characters and the bpp. Optionally you can enumerate the characters to include only them into the final font. To use the generated font declare it with _LV_FONT_DECLARE(my_font_name)_. After that, the font can be used as the built-in fonts.

Font example

aeuois
äéüöíß

Right ›

```
/*Create a new style for the label*/
static lv_style_t style;
lv_style_copy(&style, &lv_style_plain);
style.text.color = LV_COLOR_BLUE;
style.text.font = &lv_font_dejavu_40;    /*Unicode and symbol fonts already assigned by the library*/

lv_obj_t *label;

/*Use ASCII and Unicode letters*/
label = lv_label_create(lv_scr_act(), NULL);
lv_obj_set_pos(label, 20, 20);
lv_label_set_style(label, &style);
lv_label_set_text(label, "aeuois\n"
                         "äéüöíß");

/*Mix text and symbols*/
label = lv_label_create(lv_scr_act(), NULL);
lv_obj_set_pos(label, 20, 100);
lv_label_set_style(label, &style);
lv_label_set_text(label, "Right "SYMBOL_RIGHT);
```

Animasyon

Written for v5.1

You can automatically change the value (animate) of a variable between a start and an end value using an **animator function** with `void func(void* var,int32_t value)` prototype. The animation will happen by the periodical calling of the animator function with the corresponding value parameter.

To **create an animation** you have to initializes an _lv_anim_t_ variable (there is a template in lv_anim.h):

```
lv_anim_t a;
a.var = button1;                        /*Variable to animate*/
a.start = 100;                           /*Start value*/
a.end = 300;                            /*End value*/
a.fp = (lv_anim_fp_t)lv_obj_set_height;                        /*Function to be used to animate*/
a.path = lv_anim_path_linear;                             /*Path of animation*/
a.end_cb = NULL;                          /*Callback when the animation is ready*/
a.act_time = 0;                            /*Set < 0 to make a delay [ms]*/
a.time = 200;                          /*Animation length [ms]*/
a.playback = 0;                            /*1: animate in reverse direction too when the normal is ready*/
a.playback_pause = 0;                            /*Wait before playback [ms]*/
a.repeat = 0;                          /*1: Repeat the animation (with or without playback)*/
a.repeat_pause = 0;                           /*Wait before repeat [ms]*/

lv_anim_create(&a);                     /*Start the animation*/
```

The `anim_create(&a)` will register the animation and immediately **applies the *start*** value regardless to the set delay.

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

- **lv_anim_path_linear** linear animation

- **lv_anim_path_step** change in one step at the end

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.

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_ end _lv_obj_set_y_). But only one animation can exist with a given variable and function pair. Therefore the _lv_anim_create()_ function will delete the already existing variable-function animations.

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

Kodlama Kılavuzu

Revision 2

File format

Use lv_misc/lv_templ.c and lv_misc/lv_templ.h

Naming conventions

- Words are separated by '_'

- In variable and function names use only lower case letters (e.g. *height_tmp*)

- In enums and defines use only upper case letters (e.g. *e.g. MAX_LINE_NUM*)

- Global names (API):
  - starts with *lv*

  - followed by module name: *btn*, *label*, *style* etc.

  - followed by the action (for functions): *set*, *get*, *refr* etc.

  - closed with the subject: *name*, *size*, *state* etc.

- Typedefs
  - prefer `typedef struct` and `typedef enum` instead of `struct name` and `enum name`

  - always end `typedef struct` and `typedef enum` type names with `_t`

- Abbreviations:
  - Use abbreviations on public names only if they become longer than 32 characters

  - Use only very straightforward (e.g. pos: position) or well-established (e.g. pr: press) abbreviations

Coding guide

- Functions:
  - Try to write function shorter than is 50 lines

  - Always shorter than 100 lines (except very straightforwards)

- Variables:
  - One line, one declaration (BAD: char x, y;)

  - Use `<stdint.h>` (*uint8_t*, *int32_t* etc)

  - Declare variables when needed (not all at function start)

  - Use the smallest required scope

  - Variables in a file (outside functions) are always *static*

  - Do not use global variables (use functions to set/get static variables)

Comments

Before every function have a comment like this:

```
/**
 * Return with the screen of an object
 * @param obj pointer to an object
 * @return pointer to a screen
 */
lv_obj_t * lv_obj_get_scr(lv_obj_t * obj);
```

Always use `/* Something */` format and NOT `//Something`

Write readable code to avoid descriptive comments like: `x++; /* Add 1 to x */` . The code should show clearly what you are doing.

You should write **why** have you done this: `x++; /*Because of closing '\0' of the string */`

Short "code summaries" of a few lines are accepted. E.g. `/*Calculate the new coordinates*/`

In comments use `` `` `` when referring to a variable. E.g. `/*Update the value of `x_act`*/`

Formatting

Here is example to show bracket placing and using of white spaces:

```
/**
 * Set a new text for a label. Memory will be allocated to store the text by the label.
 * @param label pointer to a label object
 * @param text '\0' terminated character string. NULL to refresh with the current text.
 */
void lv_label_set_text(lv_obj_t * label, const char * text)
{   /* Main brackets of functions in new line*/

    if(label == NULL) return; /*No bracket only if the command is inline with the if statement*/

    lv_obj_inv(label);

    lv_label_ext_t * ext = lv_obj_get_ext(label);

    /*Comment before a section */
    if(text == ext->txt || text == NULL) {  /*Bracket of statements start inline*/
        lv_label_refr_text(label);
        return;
    }

    ...
}
```

Use 4 spaces indentation instead of tab.

You can use **astyle** to format the code. The required config flies are: `docs/astyle_c` and `docs/astyle_h` . To format the source files: `$ find . -type f -name "*.c" | xargs astyle --options=docs/astyle_c`

To format the header files: `$ find . -type f -name "*.h" | xargs astyle --options=docs/astyle_h`

Append `-n` to the end to skip creation of backup file OR use `$ find . -type f -name "*.bak" -delete` (for source file's backups) and `find . -type f -name "*.orig" -delete` (for header file's backups)

Anahtar (lv_sw)

Written for v5.1, revision 2

Overview

The Switch can be used to **turn on/off** something. The look like a little slider. The state of the switch can be changed by:

- Clicking on it

- Sliding it

- Using `lv_sw_on(sw)` and `lv_sw_off(sw)` functions

A **callback function** can be assigned to call when the user uses the switch: `lv_sw_set_action(sw, my_action)` .

New in v5.3: Switches can be animated by calling `lv_sw_set_anim_time(sw, anim_ms)` .

Style usage

You can modify the Switch's styles with `lv_sw_set_style(sw, LV_SW_STYLE_..., &style)` .

- **LV_SW_STYLE_BG** Style of the background. All *style.body* properties are used. The *padding* values make the Switch smaller then the knob. (negative value makes is larger)

- **LV_SW_STYLE_INDIC** Style of the indicator. All *style.body* properties are used. The *padding* values make the indicator smaller then the background.

- **LV_SW_STYLE_KNOB_OFF** Style of the knob when the switch is off. The *style.body* properties are used except padding.

- **LV_SW_STYLE_KNOB_ON** Style of the knob when the switch is on. The *style.body* properties are used except padding.

Notes

- The Knob is not a real object it is only drawn above the Bar

Example

```c
/*Create styles for the switch*/
static lv_style_t bg_style;
static lv_style_t indic_style;
static lv_style_t knob_on_style;
static lv_style_t knob_off_style;
lv_style_copy(&bg_style, &lv_style_pretty);
bg_style.body.radius = LV_RADIUS_CIRCLE;

lv_style_copy(&indic_style, &lv_style_pretty_color);
indic_style.body.radius = LV_RADIUS_CIRCLE;
indic_style.body.main_color = LV_COLOR_HEX(0x9fc8ef);
indic_style.body.grad_color = LV_COLOR_HEX(0x9fc8ef);
indic_style.body.padding.hor = 0;
indic_style.body.padding.ver = 0;

lv_style_copy(&knob_off_style, &lv_style_pretty);
knob_off_style.body.radius = LV_RADIUS_CIRCLE;
knob_off_style.body.shadow.width = 4;
knob_off_style.body.shadow.type = LV_SHADOW_BOTTOM;

lv_style_copy(&knob_on_style, &lv_style_pretty_color);
knob_on_style.body.radius = LV_RADIUS_CIRCLE;
knob_on_style.body.shadow.width = 4;
knob_on_style.body.shadow.type = LV_SHADOW_BOTTOM;

/*Create a switch and apply the styles*/
lv_obj_t *sw1 = lv_sw_create(lv_scr_act(), NULL);
lv_sw_set_style(sw1, LV_SW_STYLE_BG, &bg_style);
lv_sw_set_style(sw1, LV_SW_STYLE_INDIC, &indic_style);
lv_sw_set_style(sw1, LV_SW_STYLE_KNOB_ON, &knob_on_style);
lv_sw_set_style(sw1, LV_SW_STYLE_KNOB_OFF, &knob_off_style);
lv_obj_align(sw1, NULL, LV_ALIGN_CENTER, 0, -50);

/*Copy the first switch and turn it ON*/
lv_obj_t *sw2 = lv_sw_create(lv_scr_act(), sw1);
lv_sw_set_on(sw2);
lv_obj_align(sw2, NULL, LV_ALIGN_CENTER, 0, 50);
```

Açılır liste (lv_ddlist)

Written for v5.3, revision 2

Overview

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

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

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

A **callback function** can be specified with `lv_ddlist_set_action(ddlist, my_action)` to call when a new option is selected.

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

The **width** is also adjusted automatically. To prevent this apply `lv_ddlist_set_hor_fit(ddlist, false)` and set the width manually by `lv_obj_set_width(ddlist, width)`.

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

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

New in v5.3 is the ability to enable an arrow on the side of the drop down list. To use this feature you can call `lv_ddlist_set_draw_arrow(ddlist, true)`.

Style usage

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

- **LV_DDLIST_STYLE_BG** Style of the background. All *style.body* properties are used. It is used for the label's style from *style.text*. Default: _lv_style_pretty_

- **LV_DDLIST_STYLE_SEL** Style of the selected option. The *style.body* properties are used. The selected option will be recolored with *text.color*. Default: _lv_style_plain_color_

- **LV_DDLIST_STYLE_SB** Style of the scrollbar. The *style.body* properties are used. Default: _lv_style_plain_color_

Example

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

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

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

.
.
.

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

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

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

Buton (lv_btn)

Written for v5.1(rev.2)

Genel Bakış

Butonlar kullanıcının **basmak**, **bırakmak** veya **uzun basmak** eylemlerine bildiri fonksiyoları (`lv_action_t` fonksiyon işaretçileri) ile reaksiyon gösterir. Bildiri fonksiyonu `lv_btn_set_action(btn, ACTION_TYPE, bildiri_fonk)` ile ayarlanır. Olası aksiyon türleri:

- LV_BTN_ACTION_CLICK: Buton basıldıktan sonra bırakıldı. (tıklandı)

- LV_BTN_ACTION_PR: Buton'a basıldı

- LV_BTN_ACTION_LONG_PR: Buton'a uzun basıldı

- LV_BTN_ACTION_LONG_PR_REPEAT: Buton'a uzun süre basıldı ve bu aksiyon belirli aralıklarla tetiklendi

Buttons can be in one of the **five possible states**: Butonlar **beş olası durumlar** dan birinde bulunabilir:

- LV_BTN_STATE_REL Bırakma durumu

- LV_BTN_STATE_PR Basma durumu

- LV_BTN_STATE_TGL_REL Bırakma surumuna geç (Dorum açık)

- LV_BTN_STATE_TGL_PR Basma durumuna geç (Basma durumu açık)

- LV_BTN_STATE_INA Pasif durum

Butonlar `lv_btn_set_toggle(btn, true)` fonksiyonu ile **durum butonu** olarak ayarlanabilir. Bu durumda bırakma, buton bırakma durumuna geçer.

Manuel olarak buton durumunu `lv_btn_set_state(btn, LV_BTN_STATE_TGL_REL)` fonksiyonu ile ayarlanabilir.

Bir buton **Pasif duruma** sadece manuel olarak (_lv_btn_set_state()_) fonksiyonu ile geçebilir. Pasif durumda hiçbir aksiyon çağırılamaz.

[Konteyner](#)'a benzer olarak butonlarda **katman** ve **otomatik sığdır** özellikeri vardır:

- `lv_btn_set_layout(btn, LV_LAYOUT_...)` fonksiyonu ile katman ayarlanır. Varsayılan katman LV_LAYOUT_CENTER'dır. Bu sebepten eğer etiket(label) eklenirse, etiket otomatik olarak ortaya yerleşir.

- `lv_btn_set_fit(btn, hor_en, ver_en)` fonksiyonu butonun genişlik ve yüksekliğini alt birimlere bağlı olarak otomatik aktifleştirir.

Biçim kullanımı

Buton 5 durum için 5 bağımsız biçime sahiptir. `lv_btn_set_style(btn, LV_BTN_STYLE_..., &style)` fonksiyonu ile ayarlanır. Biçimler *style.body* özellirlerini kullanır.

- **LV_BTN_STYLE_REL** Bırakma durum biçimi. Varsayılan: _lv_style_btn_rel_

- **LV_BTN_STYLE_PR** Basma durum biçimi. Varsayılan: _lv_style_btn_pr_

- **LV_BTN_STYLE_TGL_REL** Bırak geç durum biçimi. Varsayılan: _lv_style_btn_tgl_rel_

- **LV_BTN_STYLE_TGL_PR** Bas geç durum biçimi. Varsayılan: _lv_style_btn_tgl_pr_

- **LV_BTN_STYLE_INA** Pasif durum biçimi. Varsayılan: _lv_style_btn_ina_

Notlar

- Eğer buton sürüklenirse, butonun tıklama ve uzun basma aksiyonları çağırılamaz

- Eğer buton uzun basıldığında ve butonun uzun basılma aksiyonu ayarlanmış ise butonun tıklama aksiyonu çağırılamaz

Örneğin

Default buttons

Normal

Toggled

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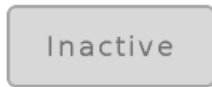
```c
static lv_res_t btn_click_action(lv_obj_t * btn)
{
    uint8_t id = lv_obj_get_free_num(btn);

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

    /* Buton bırakıldı.
     * Birşeyleri burada yap */

    return LV_RES_OK; /*Return OK Eğer buton silinmediyse*/
}

.
.
.

/*Etiket başlığı oluştur*/
lv_obj_t * label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label, "Default buttons");
lv_obj_align(label, NULL, LV_ALIGN_IN_TOP_MID, 0, 5);

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

/*Buton'a etiket ekle*/
label = lv_label_create(btn1, NULL);
lv_label_set_text(label, "Normal");

/*Butonu kopyala ve değiştir(toggle) durumuna ayarla. (Bırakma aksiyonuda kopyalandı)*/
lv_obj_t * btn2 = lv_btn_create(lv_scr_act(), btn1);
lv_obj_align(btn2, btn1, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
lv_btn_set_state(btn2, LV_BTN_STATE_TGL_REL);  /*Değiştir durumunu ayarla*/
lv_obj_set_free_num(btn2, 2);              /*Buton için tek-eşsiz sayı ayarla*/

/*Değiştir(toggle) butonuna etiket ekle*/
label = lv_label_create(btn2, NULL);
lv_label_set_text(label, "Toggled");

/*Butonu kopyala ve pasif durumu aktifleştir.*/
lv_obj_t * btn3 = lv_btn_create(lv_scr_act(), btn1);
lv_obj_align(btn3, btn2, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
lv_btn_set_state(btn3, LV_BTN_STATE_INA);   /*Pasif durumu ayarla*/
lv_obj_set_free_num(btn3, 3);              /*Buton için tek-eşsiz sayı ayarla*/

/*Pasif butona etiket ekle*/
label = lv_label_create(btn3, NULL);
lv_label_set_text(label, "Inactive");
```

Çizge (lv_chart)

Written for v5.1

Overview

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

You have several options to set the data of series:

1. Set the values manually in the array like `ser1->points[3] = 7` and refresh the chart with `lv_chart_refresh(chart)`.

2. Use the `lv_chart_set_next(chart, ser, value)` function to shift all data to left and set a new data on the most right position.

3. Initialize all points to a given value with: `lv_chart_init_points(chart, ser, value)`.

4. Set all points from an array with: `lv_chart_set_points(chart, ser, value_array)`.

There are four **data display types**:

- LV_CHART_TYPE_NONE: do not display the points. It can be used if you would like to add your own draw method.

- LV_CHART_TYPE_LINE: draw lines between the points

- LV_CHART_TYPE_COL: Draw columns

- LV_CHART_TYPE_POINT: Draw points

You can specify the display type with `lv_chart_set_type(chart, TYPE)`. The `LV_CHART_TYPE_LINE | LV_CHART_TYPE_POINT` type is also valid to draw both lines and points.

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

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

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

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

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

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

Style usage

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

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

The series related parameters can be set directly for each chart with `lv_chart_set_series_width()`, `lv_chart_set_series_opa()` and `lv_chart_set_series_dark()`.

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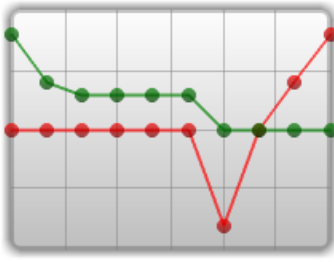
```
/*Create a style for the chart*/
static lv_style_t style;
lv_style_copy(&style, &lv_style_pretty);
style.body.shadow.width = 6;
style.body.shadow.color = LV_COLOR_GRAY;
style.line.color = LV_COLOR_GRAY;

/*Create a chart*/
lv_obj_t * chart;
chart = lv_chart_create(lv_scr_act(), NULL);
lv_obj_set_size(chart, 200, 150);
lv_obj_set_style(chart, &style);
lv_obj_align(chart, NULL, LV_ALIGN_CENTER, 0, 0);
lv_chart_set_type(chart, LV_CHART_TYPE_POINT | LV_CHART_TYPE_LINE);   /*Show lines and points too*/
lv_chart_set_series_opa(chart, LV_OPA_70);                           /*Opacity of the data series*/
lv_chart_set_series_width(chart, 4);                                 /*Line width and point radious*/

lv_chart_set_range(chart, 0, 100);

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

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

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

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

Çizgi (lv_line)

Written for v5.1

Overview

The line object is capable of **drawing straight lines** between a set of points. The points has to be stored in an `lv_point_t` array and passed to the object by the `lv_line_set_points(lines, point_array, point_num)` function.

It is possible to **automatically set the size** of the line object according to its points. You can enable it with the `lv_line_set_auto_size(line, true)` function. If enabled then when the points are set then the object width and height will be changed according to the max. x and max. y coordinates among the points. The *auto size* is enabled by default.

Basically the *y == 0* point is in the top of the object but you can **invert the y coordinates** with `lv_line_set_y_invert(line, true)`. After it the y coordinates will be subtracted from object's height.

Style usage

- **style.line** properties are used

Notes

Example

```
/*Create an array for the points of the line*/
static lv_point_t line_points[] = { {5, 5}, {70, 70}, {120, 10}, {180, 60}, {240, 10} };

/*Create line with default style*/
lv_obj_t * line1;
line1 = lv_line_create(lv_scr_act(), NULL);
lv_line_set_points(line1, line_points, 5);      /*Set the points*/
lv_obj_align(line1, NULL, LV_ALIGN_IN_TOP_MID, 0, 20);

/*Create new style (thin light blue)*/
static lv_style_t style_line2;
lv_style_copy(&style_line2, &lv_style_plain);
style_line2.line.color = LV_COLOR_MAKE(0x2e, 0x96, 0xff);
style_line2.line.width = 2;

/*Copy the previous line and apply the new style*/
lv_obj_t * line2 = lv_line_create(lv_scr_act(), line1);
lv_line_set_style(line2, &style_line2);
lv_obj_align(line2, line1, LV_ALIGN_OUT_BOTTOM_MID, 0, -20);

/*Create new style (thick dark blue)*/
static lv_style_t style_line3;
lv_style_copy(&style_line3, &lv_style_plain);
style_line3.line.color = LV_COLOR_MAKE(0x00, 0x3b, 0x75);
style_line3.line.width = 5;

/*Copy the previous line and apply the new style*/
lv_obj_t * line3 = lv_line_create(lv_scr_act(), line1);
lv_line_set_style(line3, &style_line3);
lv_obj_align(line3, line2, LV_ALIGN_OUT_BOTTOM_MID, 0, -20);
```

Etiket (lv_label)

Written for v5.1

Overview

The Labels are the basic objects to **display text**. There is no limitation in the text size because it's stored dynamically. You can modify the text in runtime at any time with lv_label_set_text().

You can use `\n` to make line break. For example: `"line1\nline2\n\nline4"`

The size of the label object can be automatically expanded to the text size or the text can be manipulated according to several **long mode policies**:

- LV_LABEL_LONG_EXPAND: Expand the object size to the text size

- LV_LABEL_LONG_BREAK: Keep the object width, break (wrap) the too long lines and expand the object height

- LV_LABEL_LONG_DOTS: Keep the object size, break the text and write dots in the last line

- LV_LABEL_LONG_SCROLL: Expand the object size and scroll the text on the parent (move the label object)

- LV_LABEL_LONG_ROLL: Keep the size and roll just the text (not the object)

You can specify the long mode with: `lv_label_set_long_mode(label, long_mode)`

Labels are able to show text from a **static array**. Use: `lv_label_set_static_text(label, char_array)` . In this case, the text is not stored in the dynamic memory but the given array is used instead. Keep in my the array can't be a local variable which destroys when the function exits.

You can also use a **raw character array** as label text. The array doesn't have to be `\0` terminated. In this case, the text will be saved to the dynamic memory. To set a raw character array use the `lv_label_set_array_text(label, char_array)` function.

The label's **text can be aligned** to the left, right or middle with `lv_label_set_align(label, LV_LABEL_ALIGN_LEFT/RIGHT/CENTER)`

You can enable to **draw a background** for the label with `lv_label_set_body_draw(label, draw)`

In the text, you can use commands to **re-color parts of the text**. For example: `"Write a #ff0000 red# word"` . This feature can be enabled individually for each label by `lv_label_set_recolor()` function.

The labels can display symbols besides letters. Learn more about symbols [here](#).

The labels' **default style** is `NULL` so they inherit the parent's style.

Style usage

- Use all properties from `style.text`

- For background drawing `style.body properties` are used

Notes

The label's **click enable attribute is disabled** by default. You can enable clicking with `lv_obj_set_click(label, true)`

Example

Title Label

Align lines to the middle

Re-color words of the text

If a line become too long it
can be automatically broken
into multiple lines

```c
/*Create label on the screen. By default it will inherit the style of the screen*/
lv_obj_t * title = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(title, "Title Label");
lv_obj_align(title, NULL, LV_ALIGN_IN_TOP_MID, 0, 20);  /*Align to the top*/

/*Create anew style*/
static lv_style_t style_txt;
lv_style_copy(&style_txt, &lv_style_plain);
style_txt.text.font = &lv_font_dejavu_40;
style_txt.text.letter_space = 2;
style_txt.text.line_space = 1;
style_txt.text.color = LV_COLOR_HEX(0x606060);

/*Create a new label*/
lv_obj_t * txt = lv_label_create(lv_scr_act(), NULL);
lv_obj_set_style(txt, &style_txt);                      /*Set the created style*/
lv_label_set_long_mode(txt, LV_LABEL_LONG_BREAK);       /*Break the long lines*/
lv_label_set_recolor(txt, true);                        /*Enable re-coloring by commands in the text*/
lv_label_set_align(txt, LV_LABEL_ALIGN_CENTER);         /*Center aligned lines*/
lv_label_set_text(txt, "Align lines to the middle\n\n"
                       "#000080 Re-color# #0000ff words of# #6666ff the text#\n\n"
                       "If a line become too long it can be automatically broken into multiple lines");
lv_obj_set_width(txt, 300);                             /*Set a width*/
lv_obj_align(txt, NULL, LV_ALIGN_CENTER, 0, 20);        /*Align to center*/
```

Kaydırıcı (lv_slider)

Written for v5.1

Overview

The Slider object looks like a **Bar** supplemented **with a Knob**. The Knob can be **dragged to set a value**. The Slider also can be vertical or horizontal.

To set an **initial value** use `lv_slider_set_value(slider, new_value)` function or `lv_slider_set_value_anim(slider, new_value, anim_time)` to set the value with an animation.

To specify the **range** (min, max values) the `lv_slider_set_range(slider, min , max)` can be used.

A **callback function** can be assigned to call when a new value is set by the user: `lv_slider_set_action(slider, my_action)`.

The **knob can be placed** two ways:

- inside the background on min/max values

- on the edges on min/max values

Use the `lv_slider_set_knob_in(slider, true/false)` to choose between the modes. (*knob_in == false* is the default)

Style usage

You can modify the slider's styles with `lv_slider_set_style(slider, LV_SLIDER_STYLE_..., &style)`.

- **LV_SLIDER_STYLE_BG** Style of the background. All *style.body* properties are used. The *padding* values make the slider smaller then the knob. (negative value makes is larger)

- **LV_SLIDER_STYLE_INDIC** Style of the indicator. All *style.body* properties are used. The *padding* values make the indicator smaller then the background.

- **LV_SLIDER_STYLE_KNOB** Style of the knob. The *style.body* properties are used except padding

Notes

- The Knob is not a real object it is only drawn above the Bar

Example

```c
/*Called when a new value id set on the slider*/
static lv_res_t slider_action(lv_obj_t * slider)
{
    printf("New slider value: %d\n", lv_slider_get_value(slider));

    return LV_RES_OK;
}


.
.
.

/*Create a default slider*/
lv_obj_t * slider1 = lv_slider_create(lv_scr_act(), NULL);
lv_obj_set_size(slider1, 160, 30);
lv_obj_align(slider1, NULL, LV_ALIGN_IN_TOP_RIGHT, -30, 30);
lv_slider_set_action(slider1, slider_action);
lv_bar_set_value(slider1, 70);


/*Create a label right to the slider*/
lv_obj_t * slider1_label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(slider1_label, "Default");
lv_obj_align(slider1_label, slider1, LV_ALIGN_OUT_LEFT_MID, -20, 0);

/*Create a bar, an indicator and a knob style*/
static lv_style_t style_bg;
static lv_style_t style_indic;
static lv_style_t style_knob;

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

lv_style_copy(&style_indic, &lv_style_pretty);
style_indic.body.grad_color =  LV_COLOR_GREEN;
style_indic.body.main_color =  LV_COLOR_LIME;
style_indic.body.radius = LV_RADIUS_CIRCLE;
style_indic.body.shadow.width = 10;
style_indic.body.shadow.color = LV_COLOR_LIME;
style_indic.body.padding.hor = 3;
style_indic.body.padding.ver = 3;

lv_style_copy(&style_knob, &lv_style_pretty);
style_knob.body.radius = LV_RADIUS_CIRCLE;
style_knob.body.opa = LV_OPA_70;
style_knob.body.padding.ver = 10 ;

/*Create a second slider*/
lv_obj_t * slider2 = lv_slider_create(lv_scr_act(), slider1);
lv_slider_set_style(slider2, LV_SLIDER_STYLE_BG, &style_bg);
lv_slider_set_style(slider2, LV_SLIDER_STYLE_INDIC,&style_indic);
lv_slider_set_style(slider2, LV_SLIDER_STYLE_KNOB, &style_knob);
lv_obj_align(slider2, slider1, LV_ALIGN_OUT_BOTTOM_MID, 0, 30); /*Align below 'bar1'*/

/*Create a second label*/
lv_obj_t * slider2_label = lv_label_create(lv_scr_act(), slider1_label);
lv_label_set_text(slider2_label, "Modified");
lv_obj_align(slider2_label, slider2, LV_ALIGN_OUT_LEFT_MID, -30, 0);
```

Klavye (lv_kb)

Written for v5.1

Overview

As it names shows the **Keyboard** object provides a keyboard to **write text**. You can assign a Text area to the Keyboard to put the clicked characters there. To assign the Text area use `lv_kb_set_ta(kb, ta)` .

The keyboard contains an _Ok_ and a *Hide* button. An ok and a hide action can be specified by `lv_kb_set_ok_action(kb, action)` and `lv_kb_set_hide_action(kb, action)` to add callbacks to Ok/Hide clicks. If no action is specified then the buttons will delete the Keyboard.

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 an the new's will be shown. Clicking on _Ok_ or *Hide* will also hide the cursor. The cursor manager feature is enabled by `lv_kb_set_cursor_manage(kb, true)` . The default is not manage.

The Keyboards have two **modes**:

- LV_KB_MODE_TEXT: display letters, number and special characters

- LV_KB_MODE_NUM: display numbers, +/- sign and dot

To set the mode use `lv_kb_set_mode(kb, mode)` . The default is _LV_KB_MODE_TEXT_

You can specify a **new map** (layout) for the keyboard with `lv_kb_set_map(kb, map)` . It works like a the Button matrix so control character can be added to the layout the set button width and other attributes. Keep in mind using following keywords will have the same effect as with the original map: _SYMBOL_OK_, _SYMBOL_CLOSE_, _SYMBOL_LEFT_, _SYMBOL_RIGHT_, *ABC*, *abc*, *Enter*, *Del*, _#1_, +/- .

Style usage

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

- **LV_KB_STYLE_BG** Background style. Uses all *style.body* properties including *padding* Default: _lv_style_pretty_

- **LV_KB_STYLE_BTN_REL** style of the released buttons. Default: _lv_style_btn_rel_

- **LV_KB_STYLE_BTN_PR** style of the pressed buttons. Default: _lv_style_btn_pr_

- **LV_KB_STYLE_BTN_TGL_REL** style of the toggled released buttons. Default: _lv_style_btn_tgl_rel_

- **LV_KB_STYLE_BTN_TGL_PR** style of the toggled pressed buttons. Default: _lv_style_btn_tgl_pr_

- **LV_KB_STYLE_BTN_INA** style of the inactive buttons. Default: _lv_style_btn_ina_

Example
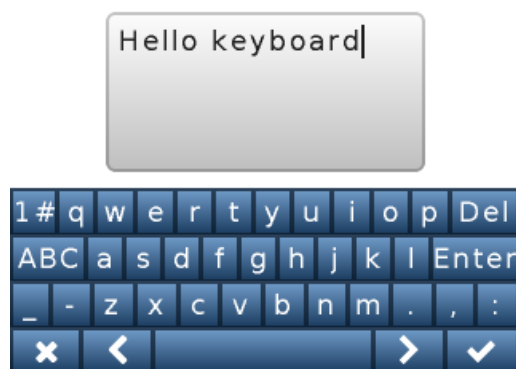
```
/*Create styles for the keyboard*/
static lv_style_t rel_style, pr_style;

lv_style_copy(&rel_style, &lv_style_btn_rel);
rel_style.body.radius = 0;

lv_style_copy(&pr_style, &lv_style_btn_pr);
pr_style.body.radius = 0;

/*Create a keyboard and apply the styles*/
lv_obj_t *kb = lv_kb_create(lv_scr_act(), NULL);
lv_kb_set_cursor_manage(kb, true);
lv_kb_set_style(kb, LV_KB_STYLE_BG, &lv_style_transp_tight);
lv_kb_set_style(kb, LV_KB_STYLE_BTN_REL, &rel_style);
lv_kb_set_style(kb, LV_KB_STYLE_BTN_PR, &pr_style);

/*Create a text area. The keyboard will write here*/
lv_obj_t *ta = lv_ta_create(lv_scr_act(), NULL);
lv_obj_align(ta, NULL, LV_ALIGN_IN_TOP_MID, 0, 10);
lv_ta_set_text(ta, "");

/*Assign the text area to the keyboard*/
lv_kb_set_ta(kb, ta);
```

Konteynır (lv_cont)

Written for v5.1 (rev.3)

Genel Bakış

Konteynerler bazı özel özellirleri ile **dikdörtgen'e benzer nesneler** dir.

Konteynerlara **katman** uygulandığında alt birimlerede uygulanmış olur. Katman aralığı(mesafe) style.body.padding.hor/ver/inner özelliği ile yapılır. Olası katman seçenekleri:

- LV_LAYOUT_OFF: Alt birimleri hizalama(olumsuz)

- LV_LAYOUT_CENTER: Alt birimleri merkezde sutun şeklinde hizala ve aralarında *pad.inner* boşluk bırak

- LV_LAYOUT_COL_L: Alt birimleri soldaki bir sütunda hizalayın. Solda *pad.hor* alanı, üstte *pad.ver* alanı ve alt birimler arasında *pad.inner* alanı olsun.

- LV_LAYOUT_COL_M: Alt birimleri ortalanmış sütunla hizalayın. En üstte *pad.ver* alanını ve alt birimler arasında *pad.inner* alanını tutun.

- LV_LAYOUT_COL_R: Alt birimleri sağa dayalı bir sütunda hizalayın. Sağ tarafta *pad.hor* alanı, üstte *pad.ver* alanı ve alt birimler arasında *pad.inner* alanı olsun.

- LV_LAYOUT_ROW_T: Alt birimleri en üst sıradaki bir satırda hizalayın. Solda *pad.hor* alanı, üstte *pad.ver* alanı ve alt birimler arasında *pad.inner* alanı olsun.

- LV_LAYOUT_ROW_M: Orta sıradaki alt birimleri hizalayın. Sol tarafta *pad.hor* alanı ve alt birimler arasında *pad.inner* boşluk bulundurun.

- LV_LAYOUT_ROW_B: Alttaki bir satırdaki alt birimleri hizalayın. Soldaki *pad.hor* alanını, alt kısımdaki *pad.ver* alanını ve alt birirmler arasında *pad.inner* alanı tutun.

- LV_LAYOUT_PRETTY: Bir satırda olabildiğince çok nesne koyun (en azından *pad.inner* boşluk ve yanlarda *pad.hor* boşluk ile) ve yeni bir sıraya başlayın. Her bir çizgideki alanı, alt birimler arasında eşit olarak bölün. Üst kısımda *pad.ver* alanını ve satırların arasındaki *pad.inner* alanını tutun.

- LV_LAYOUT_GRID: PRETTY LAYOUT'a benzer, ancak yatay alanı eşit olarak bölmesin, sadece *pad.hor* boşluğa izin verin.

Tüm alt birimleri dahil etmek için kapsayıcı boyutunu otomatik olarak ayarlayan bir **otomatik sığdır(uyum)** özelliğini etkinleştirebilirsiniz. Solda ve sağda *pad.hor* alanı ve altta *pad.ver* alanı olacak. Otomatik yerleştirme, `lv_cont_set_fit(cont, true, true)` işleviyle yatay, dikey veya her iki yönde de etkinleştirilebilir. İkinci parametre yataydır, üçüncü parametre dikey sığdırma etkinliğidir.

Biçim Kullanımı

- **style.body** özellikleri kullanılır.

Notlar

- **Alt birim pozisyonunu hor / ver fit etkin iken ayarlayamazsınız**. Ne olduğunu hayal edelim. Uyum sağlandığında tek alt birimin pozisyonunu değiştirirseniz, konteyner yeni pozisyonu "etrafında" hareket ettirir / yerleştirir. Bir konteynerda daha fazla nesneniz varsa, bunları birbirine göre hizalayabilirsiniz. Bir geçici çözüm olarak konteyner üzerinde küçük saydam bir nesne oluşturabilirsiniz. Alt birimleri "takip etmemek" için konteyner'ı düzeltecektir.

Örnek

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

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

lv_obj_align(box1, NULL, LV_ALIGN_IN_TOP_LEFT, 10, 10);        /*Konteyner'ı hizala*/

/*Biçim oluştur*/
static lv_style_t style;
lv_style_copy(&style, &lv_style_pretty_color);
style.body.shadow.width = 6;
style.body.padding.hor = 5;                                    /*Büyük bir yatay boşluk oluştur*/

/*Başka bir konteyner oluştur*/
lv_obj_t * box2;
box2 = lv_cont_create(lv_scr_act(), NULL);
lv_obj_set_style(box2, &style);        /*Yeni biçim ayarla*/
lv_cont_set_fit(box2, true, false); /*Dikey sığdırmayı aktif etme*/
lv_obj_set_height(box2, 55);           /*Sabit yükseklik ayarla*/

/*Yeni konteyner'a metin ekle*/
lv_obj_t * txt2 = lv_label_create(box2, NULL);
lv_label_set_text(txt2, "No vertical fit 1...\n"
                        "No vertical fit 2...\n"
                        "No vertical fit 3...\n"
                        "No vertical fit 4...");

/*Konteyner'ı öncekinin altına hizala*/
lv_obj_align(box2, box1, LV_ALIGN_OUT_BOTTOM_MID, 30, -30);
```

LED (lv_led)

Written for v5.1

Overview

The LEDs are rectangle-like (or circle) object. You can set their **brightness** with `lv_led_set_bright(led, bright)`. The brightness should be between 0 (darkest) and 255 (lightest).

Use `lv_led_on(led)` and `lv_led_off(led)` to set the brightness to a predefined ON or OFF value. The `lv_led_toggle(led)` toggles between the ON and OFF state.

Style usage

The LED uses one style which can be set by `lv_led_set_style(led, &style)`. To determine the appearance the **style.body** properties are used. The colors are darkened and shadow width is reduced at a lower brightness and gains its original value at brightness 255 to show a lighting effect. The default style is: `lv_style_pretty_color`.

Notes

- Typically the default style is not suitable therefore you have to create you own style. See the Examples.

Example

```
/*Create a style for the LED*/
static lv_style_t style_led;
lv_style_copy(&style_led, &lv_style_pretty_color);
style_led.body.radius = LV_RADIUS_CIRCLE;
style_led.body.main_color = LV_COLOR_MAKE(0xb5, 0x0f, 0x04);
style_led.body.grad_color = LV_COLOR_MAKE(0x50, 0x07, 0x02);
style_led.body.border.color = LV_COLOR_MAKE(0xfa, 0x0f, 0x00);
style_led.body.border.width = 3;
style_led.body.border.opa = LV_OPA_30;
style_led.body.shadow.color = LV_COLOR_MAKE(0xb5, 0x0f, 0x04);
style_led.body.shadow.width = 10;


/*Create a LED and switch it ON*/
lv_obj_t * led1  = lv_led_create(lv_scr_act(), NULL);
lv_obj_set_style(led1, &style_led);
lv_obj_align(led1, NULL, LV_ALIGN_IN_TOP_MID, 40, 40);
lv_led_on(led1);

/*Copy the previous LED and set a brightness*/
lv_obj_t * led2  = lv_led_create(lv_scr_act(), led1);
lv_obj_align(led2, led1, LV_ALIGN_OUT_BOTTOM_MID, 0, 40);
lv_led_set_bright(led2, 190);

/*Copy the previous LED and switch it OFF*/
lv_obj_t * led3  = lv_led_create(lv_scr_act(), led1);
lv_obj_align(led3, led2, LV_ALIGN_OUT_BOTTOM_MID, 0, 40);
lv_led_off(led3);

/*Create 3 labels next to the LEDs*/
lv_obj_t * label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label, "LED On");
lv_obj_align(label, led1, LV_ALIGN_OUT_LEFT_MID, -40, 0);

label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label, "LED half");
lv_obj_align(label, led2, LV_ALIGN_OUT_LEFT_MID, -40, 0);

label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label, "LED Off");
lv_obj_align(label, led3, LV_ALIGN_OUT_LEFT_MID, -40, 0);
```

Onay kutusu (lv_cb)

Overview

The Check Box objects are built from a Button **background** which contains an also Button **bullet** and a **label** to realize a classical check box. The **text** can be modified by the `lv_cb_set_text(cb, "New text")` function.

An **action** can assigned by `lv_cb_set_action(cb, action)`.

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

Style usage

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

- **LV_CB_STYLE_BG** Background style. Uses all *style.body* properties. The label's style comes from *style.text*. Default: _lv_style_transp_

- **LV_CB_STYLE_BOX_REL** Style of the released box. Uses the *style.body* properties. Default: _lv_style_btn_rel_

- **LV_CB_STYLE_BOX_PR** Style of the pressed box. Uses the *style.body* properties. Default: _lv_style_btn_pr_

- **LV_CB_STYLE_BOX_TGL_REL** Style of the checked released box. Uses the *style.body* properties. Default: _lv_style_btn_tgl_rel_

- **LV_CB_STYLE_BOX_TGL_PR** Style of the checked released box. Uses the *style.body* properties. Default: _lv_style_btn_tgl_pr_

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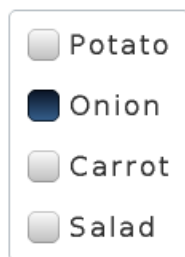
```
static lv_res_t cb_release_action(lv_obj_t * cb)
{
    /*A check box is clicked*/
    printf("%s state: %d\n", lv_cb_get_text(cb), lv_cb_is_checked(cb));

    return LV_RES_OK;
}

.
.
.


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

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

/*Create a container*/
lv_obj_t * cont;
cont = lv_cont_create(lv_scr_act(), NULL);
lv_cont_set_layout(cont, LV_LAYOUT_COL_L);       /*Arrange the children in a column*/
lv_cont_set_fit(cont, true, true);               /*Fit the size to the content*/
lv_obj_set_style(cont, &style_border);

/***************************
 * Create check boxes
 ***************************/

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

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

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

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

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

Önyükleyici (lv_preload)

Written for v5.2

Overview

The preloader object is **a spinning arc over a border**.

The **length of the arc** can be adjusted by `lv_preload_set_arc_length(preload, deg)` .

The **speed of the spinning** can be adjusted by `lv_preload_set_spin_time(preload, time_ms)` .

Style usage

The `LV_PRELOAD_STYLE_MAIN` style describes both the arc and the border style:

- **arc** is described by the `line` properties

- **border** is described by the `body.border` properties including `body.padding.hor/ver` (smaller is used) to give a smaller radius for the border.

Example


```c
/*Create a style for the Preloader*/
static lv_style_t style;
lv_style_copy(&style, &lv_style_plain);
style.line.width = 10;                    /*10 px thick arc*/
style.line.color = LV_COLOR_HEX3(0x258);       /*Blueish arc color*/

style.body.border.color = LV_COLOR_HEX3(0xBBB); /*Gray background color*/
style.body.border.width = 10;
style.body.padding.hor = 0;

/*Create a Preloader object*/
lv_obj_t * preload = lv_preload_create(lv_scr_act(), NULL);
lv_obj_set_size(preload, 100, 100);
lv_obj_align(preload, NULL, LV_ALIGN_CENTER, 0, 0);
lv_preload_set_style(preload, LV_PRELOAD_STYLE_MAIN, &style);
```

Pencere (lv_window)

Written for v5.1 (rev.4)

Genel Bakış

Pencereler, **en karmaşık** konteyner benzeri nesnelerden biridir. İki ana bölümden oluşur: Üstte bir **başlık** Konteyner ve başlığın altındaki **içerik** için bir Sayfa.

Üstbilgi **başlığı** değiştirilebilir: `lv_win_set_title(win, "Yeni Başlık")` . Başlık, üstbilginin biçimini her zaman miraslanır.

Üst bilginin sağına **kontrol butonları** `lv_win_add_btn(win, "U:/close", my_close_action)` fonksiyonu ile eklenebilir. İkinci parametre bir resim dosyası yoludur, üçüncü parametre buton bırakıldığı zaman çağrılan bir fonksiyondur. **Semboller** resim gibi kullanabilir: `lv_win_add_btn(win, SYMBOL_CLOSE, my_close_action)` .

Kontrol butonlarının boyutu `lv_win_set_btn_size(win, yeni_boyut)` fonksiyonuyla değiştirebilir.

Kaydırma çubuğunun davranışı `lv_win_set_sb_mode(win, LV_SB_MODE_...)` fonksiyonuyla ayarlanabilir.

İçerik düzenini ayarlamak için `lv_win_set_layout(win, LV_LAYOUT_...)` fonksiyonu kullanılır.

Biçim kullanımı

Pencerenin öğesine yeni bir stil ayarlamak için `lv_win_set_style(win, LV_WIN_STYLE_..., &style)` fonksiyonu kullanılır.

- **LV_WIN_STYE_BG** tüm *style.body* özelliklerini kullanan ana arka plan(üstbilgi ve içerik yer alır) (varsayılan: lv_style_plain)

- **LV_WIN_STYLE_CONTENT_BG** tüm *style.body* özelliklerini kullanan içerik sayfasının arka planı (varsayılan: lv_style_transp)

- **LV_WIN_STYLE_CONTENT_SCRL** tüm *style.body* özelliklerini kullanan içerik sayfasının kaydırma çubuğu (varsayılan: lv_style_transp)

- **LV_WIN_STYLE_SB** tüm *style.body* özelliklerini kullanan kaydırma çubuğunun biçimi. Kaydırma çubuğunun dolgusu sırasıyla yatay/dikey* dolgularıyla ayarlanır ve kaydırma çubuğunun genişliği de iç dolguyla ayarlanır. (varsayılan: lv_style_pretty_color)

- **LV_WIN_STYLE_HEADER** tüm *style.body* özelliklerini kullanan üstbilgi biçimi (varsayılan: lv_style_plain_color)

- **LV_WIN_STYLE_BTN_REL** tüm *style.body* özelliklerini kullanan butonu bırakman(üstbilgi) biçimi (varsayılan: lv_style_btn_rel)

- **LV_WIN_STYLE_BTN_PR** tüm *style.body* özelliklerini kullanan butona basma(üstbilgi) biçimi (default: lv_style_btn_pr)

Notlar

Örnek
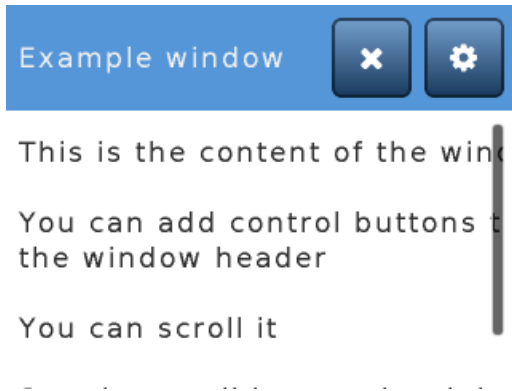
```
/*Yeni bir kaydırma çubuğu biçimi oluştur*/
static lv_style_t style_sb;
lv_style_copy(&style_sb, &lv_style_plain);
style_sb.body.main_color = LV_COLOR_BLACK;
style_sb.body.grad_color = LV_COLOR_BLACK;
style_sb.body.border.color = LV_COLOR_WHITE;
style_sb.body.border.width = 1;
style_sb.body.border.opa = LV_OPA_70;
style_sb.body.radius = LV_RADIUS_CIRCLE;
style_sb.body.opa = LV_OPA_60;

/*Yeni bir pencere oluştur*/
lv_obj_t * win = lv_win_create(lv_scr_act(), NULL);
lv_win_set_title(win, "Example window");              /*Başlığı ayarla*/
lv_win_set_style(win, LV_WIN_STYLE_SB, &style_sb);     /*Kaydırma çubuğunun biçimini ayarla*/


/*Üstbilgiye kontrol butonu ekle*/
lv_win_add_btn(win, SYMBOL_SETTINGS, my_setup_action);      /*Ayar butonu ekle*/
lv_win_add_btn(win, SYMBOL_CLOSE, lv_win_close_action);      /*Kapatma butonu ekle ve kapatma aksiyonunun kullanılması*/


/*İçerik örneği ekle*/
lv_obj_t * txt = lv_label_create(win, NULL);
lv_label_set_text(txt, "This is the content of the window\n\n"
                       "You can add control buttons to\nthe window header\n\n"
                       "You can scroll it\n\n"
                       "See the scroll bar on the right!");
```

Resim (lv_img)

Written for v5.1

Overview

The Images are the basic object to **display images**. To provide maximum flexibility the **source of the image** can be:

- a variable in the code (a C array with the pixels)

- a file stored externally (like on an SD card)

- a text with Symbols

To set the source of an image the `lv_img_set_src` function can be used.

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 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. To see how to handle external image files from LittlevGL check the Tutorial.

You can set a **symbol** from lv_symbol_def.h too. In this case, the image will be rendered as text according to the **font** specified in the style. It enables to use light weighted mono-color "letters" instead of real images. You can set symbol like this:

```
lv_img_set_src(img1, SYMBOL_OK);
```

The internal (variable) and external images support 2 **transparency handling** methods:

- **Chrome keying** `LV_COLOR_TRANSP` (lv_conf.h) will be transparent

- **Alpha byte** Add an alpha byte to every pixel

These options can be selected in the online font converter.

The images can be **re-colored in run-time** to any color according to the brightness of the pixels. It is very useful to show different states (selected, inactive, pressed etc) of an image without storing more versions of the same image. This feature can be enabled in the style by setting `img.intense` between `LV_OPA_TRANSP` (no recolor, value: 0) and `LV_OPA_COVER` (full recolor, value: 255). The default value is `LV_OPA_TRANSP` so this feature is disabled.

It is possible to **automatically set the size** of the image object to the image source's width and height if enabled by the `lv_img_set_auto_size(image, true)` function. If *auto size* is enabled then when a new file is set the object size is automatically changed. Later you can modify the size manually. If the object size is greater then the image size in any directions then the image will be repeated like a mosaic. The auto size is enabled by default if the image is not a screen.

The images' default style is NULL so they **inherit the parent's style**.

Style usage

- For images **style.img**

- For symbols **style.text**

Notes

- Symbols names (like SYMBOL_EDIT) are short strings, therefore, you can concatenate them to show more symbols.

Example

Re-color the images in run time

Use symbols from fonts as images

```
/*Declare a cogwheel image variable*/
LV_IMG_DECLARE(img_cw);

[...]

/****************************************
 * Create three images and re-color them
 ****************************************/

/*Create the first image without re-color*/
lv_obj_t * img1 = lv_img_create(lv_scr_act(), NULL);
lv_img_set_src(img1, &img_cw);
lv_obj_align(img1, NULL, LV_ALIGN_IN_TOP_LEFT, 20, 40);

/*Create style to re-color with light blue*/
static lv_style_t style_img2;
lv_style_copy( &style_img2, &lv_style_plain);
style_img2.image.color = LV_COLOR_HEX(0x003b75);
style_img2.image.intense = LV_OPA_50;

/*Create an image with the light blue style*/
lv_obj_t * img2 = lv_img_create(lv_scr_act(), img1);
lv_obj_set_style(img2, &style_img2);
lv_obj_align(img2, NULL, LV_ALIGN_IN_TOP_MID, 0, 40);

/*Create style to re-color with dark blue*/
static lv_style_t style_img3;
lv_style_copy(&style_img3, &lv_style_plain);
style_img3.image.color = LV_COLOR_HEX(0x003b75);
style_img3.image.intense = LV_OPA_90;

/*Create an image with the dark blue style*/
lv_obj_t * img3 = lv_img_create(lv_scr_act(), img2);
lv_obj_set_style(img3, &style_img3);
lv_obj_align(img3, NULL, LV_ALIGN_IN_TOP_RIGHT, -20, 40);


/****************************************
 * Create an image with symbols
 ****************************************/

/*Create a string from symbols*/
char buf[32];
sprintf(buf, "%s%s%s%s%s%s%s",
            SYMBOL_DRIVE, SYMBOL_FILE, SYMBOL_DIRECTORY, SYMBOL_SETTINGS,
            SYMBOL_POWER, SYMBOL_GPS, SYMBOL_BLUETOOTH);

/*Create style with a symbol font*/
static lv_style_t style_sym;
lv_style_copy(&style_sym, &lv_style_plain);
// The built-in fonts are extended with symbols
style_sym.text.font = &lv_font_dejavu_60;
style_sym.text.letter_space = 10;

/*Create an image and use the string as source*/
lv_obj_t * img_sym = lv_img_create(lv_scr_act(), NULL);
lv_img_set_src(img_sym, buf);
lv_img_set_style(img_sym, &style_sym);
lv_obj_align(img_sym, NULL, LV_ALIGN_IN_BOTTOM_MID, 0, -30);


/*Create description labels*/
lv_obj_t * label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label, "Re-color the images in run time");
lv_obj_align(label, NULL, LV_ALIGN_IN_TOP_MID, 0, 15);

label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label, "Use symbols from fonts as images");
lv_obj_align(label, NULL, LV_ALIGN_IN_BOTTOM_MID, 0, -80);
```

Resim buton (lv_imgbtn)

Written for v5.2

Overview

The Image button is very similar to the simple Button object. The only difference is it displays user-defined images in each state instead of drawing a button. Before reading this please learn how the Buttons work in LittelvGL: link to the button

To set the image in a state the `lv_imgbtn_set_src(imgbtn, LV_BTN_STATE_..., &img_src)` The image sources works the same as described in the Image object. TODO link

Similarly to the Button object **actions (callbacks) can be assigned** by `lv_imgbtn_set_action(imgbtn, LV_BTN_ACTION_..., my_action)`.

The **states** also work like with Button object. It can be set with `lv_imgbtn_set_state(imgbtn, LV_BTN_STATE_...)`.

The **toggle** feature can be enabled with `lv_imgbtn_set_toggle(imgbtn, true)`

Style usage

The Image buttons can have unique styles for each state. All the `style.image` properties used by the Image button:

- **image.color** Recolor the image to this color according to `intense`
- **image.intense** The extent of recoloring (0..255 or `LV_OPA_0/10/20..100`)
- **image.opa** The opacity of the object (0..255 or `LV_OPA_0/10/20..100`)

Notes

Example

```c
/*Create style to make the button darker when pressed*/
lv_style_t style_pr;
lv_style_copy(&style_pr, &lv_style_plain);
style_pr.image.color = LV_COLOR_BLACK;
style_pr.image.intense = LV_OPA_50;
style_pr.text.color = LV_COLOR_HEX3(0xaaa);

LV_IMG_DECLARE(imgbtn_green);
LV_IMG_DECLARE(imgbtn_blue);

/*Create an Image button*/
lv_obj_t * imgbtn1 = lv_imgbtn_create(lv_scr_act(), NULL);
lv_imgbtn_set_src(imgbtn1, LV_BTN_STATE_REL, &imgbtn_green);
lv_imgbtn_set_src(imgbtn1, LV_BTN_STATE_PR, &imgbtn_green);
lv_imgbtn_set_src(imgbtn1, LV_BTN_STATE_TGL_REL, &imgbtn_blue);
lv_imgbtn_set_src(imgbtn1, LV_BTN_STATE_TGL_PR, &imgbtn_blue);
lv_imgbtn_set_style(imgbtn1, LV_BTN_STATE_PR, &style_pr);          /*Use the darker style in the pressed state*/
lv_imgbtn_set_style(imgbtn1, LV_BTN_STATE_TGL_PR, &style_pr);
lv_imgbtn_set_toggle(imgbtn1, true);
lv_obj_align(imgbtn1, NULL, LV_ALIGN_CENTER, 0, -40);

/*Create a label on the Image button*/
lv_obj_t * label = lv_label_create(imgbtn1, NULL);
lv_label_set_text(label, "Button");

/*Copy the fist image button and set Toggled state*/
lv_obj_t * imgbtn2 = lv_imgbtn_create(lv_scr_act(), imgbtn1);
lv_btn_set_state(imgbtn2, LV_BTN_STATE_TGL_REL);
lv_obj_align(imgbtn2, imgbtn1, LV_ALIGN_OUT_BOTTOM_MID, 0, 20);

/*Create a label on the Image button*/
label = lv_label_create(imgbtn2, NULL);
lv_label_set_text(label, "Button");
```

Sarma çubuğu (lv_roller)

Written for v5.1 Updated to v5.2

Overview

Roller allow you to simply **select one option from more** with scrolling. Its functionalities are similar to [Drop down list](#).

The **options** are passed to the Roller as a **string** with `lv_roller_set_options(roller, options)`. The options should be separated by `\n`. For example: `"First\nSecond\nThird"`.

You can **select an option manually** with `lv_roller_set_selected(roller, id)`, where _id_ is the index of an option.

A **callback function** can be specified with `lv_roller_set_action(roller, my_action)` to call when a new option is selected.

The roller's **height** can be adjusted with `lv_roller_set_visible_row_count(roller, row_cnt)` to set number of visible options.

The **width** is adjusted automatically. To prevent this apply `lv_roller_set_hor_fit(roller, false)` and set the width manually by `lv_obj_set_width(roller, width)`. You should use `lv_roller_set_hor_fit(roller, false)` instead of `lv_cont_set_fit(lv_page_get_scrl(roller), false, false);`, ohterwise you'll get an LV_LABEL_ALIGN_LEFT style of the list label text.

The Roller's open/close **animation** time is adjusted by `lv_roller_set_anim_time(roller, anim_time)`. Zero animation time means no animation. This feature is implemented within lv_ddlist.c in v5.2: `lv_ddlist_set_anim_time(roller, anim_time);` should be used for animation.

Style usage

The `lv_roller_set_style(roller, LV_ROLLER_STYLE_..., &style)` set the styles of a roller.

- **LV_ROLLER_STYLE_BG** Style of the background. All *style.body* properties are used. It is used for the label's style from *style.text*. Gradient is applied on the top and bottom as well. Default: _lv_style_pretty_

- **LV_DDLIST_STYLE_SEL** Style of the selected option. The *style.body* properties are used. The selected option will be recolored with *text.color*. Default: _lv_style_plain_color_

Example

```c
/*Create a default roller*/
lv_obj_t *roller1 = lv_roller_create(lv_scr_act(), NULL);
lv_roller_set_options(roller1, "Apple\n"
                                "Broccoli\n"
                                "Cabbage\n"
                                "Dewberry\n"
                                "Eggplant\n"
                                "Fig\n"
                                "Grapefruit");
lv_obj_set_pos(roller1, 50, 80);


/*Create styles*/
static lv_style_t bg_style;
lv_style_copy(&bg_style, &lv_style_pretty);
bg_style.body.main_color = LV_COLOR_WHITE;
bg_style.body.grad_color = LV_COLOR_HEX3(0xddd);
bg_style.body.border.width = 0;
bg_style.text.line_space = 20;
bg_style.text.opa = LV_OPA_40;

static lv_style_t sel_style;
lv_style_copy(&sel_style, &lv_style_pretty);
sel_style.body.empty = 1;
sel_style.body.radius = LV_RADIUS_CIRCLE;
sel_style.text.color = LV_COLOR_BLUE;

/*Create a roller and apply the new styles*/
lv_obj_t *roller2 = lv_roller_create(lv_scr_act(), NULL);
lv_roller_set_options(roller2, "0\n"
                                "1\n"
                                "2\n"
                                "3\n"
                                "4\n"
                                "5\n"
                                "6\n"
                                "7\n"
                                "8\n"
                                "9");
lv_roller_set_style(roller2, LV_ROLLER_STYLE_BG, &bg_style);
lv_roller_set_selected(roller2, 3, false);
lv_roller_set_style(roller2, LV_ROLLER_STYLE_SEL, &sel_style);
lv_roller_set_visible_row_count(roller2, 3);
lv_roller_set_hor_fit(roller2, false);
lv_obj_set_width(roller2, 40);
lv_obj_set_pos(roller2, 220, 50);
```

Sekme görünümü (lv_tabview)

Written for v5.1

Overview

The Tab view object can be used to **organize content in tabs**. You can **add a new tab** with `lv_tabview_add_tab(tabview, "Tab name")` . It will return with a pointer to a Page object where you can add the tab's content.

To **select a tab** you can:

- Click on it on the header part

- Slide horizontally

- Use `lv_tabview_set_tab_act(tabview, id, anim_en)` function

The **manual sliding** can be disabled with `lv_tabview_set_sliding(tabview, false)` .

The **animation time** is adjusted by `lv_tabview_set_anim_time(tabview, anim_time)` .

A **callback function** can be assigned to **tab load** event with `lv_tabview_set_tab_load_action(tabview, action)` . The callback function need to have the following prototype:

```
void callback(lv_obj_t * tabview, uint16_t act_id);
```

Where _act_id_ means tab which will be loaded. In the action `lv_tabview_get_tab_act(tabview)` will give the id of the old tab.

Style usage

Use `lv_tabview_set_style(tabview, LV_TABVIEW_STYLE_..., &style)` to set a new style for an element of the tab view:

- **LV_TABVIEW_STYLE_BG** main background which uses all *style.body* properties (default: lv_style_plain)

- **LV_TABVIEW_STYLE_INDIC** a thin rectangle on the top to indicate the current tab. Uses all *style.body* properties. It height comes from *body.padding.inner* (default: _lv_style_plain_color_)

- **LV_TABVIEW_STYLE_BTN_BG** style of the tab buttons' background. Uses all *style.body* properties. The header height will be set automatically considering *body.padding.ver* (default: _lv_style_transp_)

- **LV_TABVIEW_STYLE_BTN_REL** style of released tab buttons. Uses all *style.body* properties. (default: _lv_style_tbn_rel_)

- **LV_TABVIEW_STYLE_BTN_PR** style of released tab buttons. Uses all *style.body* properties. (default: _lv_style_tbn_rel_)

- **LV_TABVIEW_STYLE_BTN_TGL_REL** style of toggled released tab buttons. Uses all *style.body* properties. (default: _lv_style_tbn_rel_)

- **LV_TABVIEW_STYLE_BTN_TGL_PR** style of toggled pressed tab buttons. Uses all *style.body* properties. (default: _lv_style_btn_tgl_pr_)

Notes

Example

```
/*Create a Tab view object*/
lv_obj_t *tabview;
tabview = lv_tabview_create(lv_scr_act(), NULL);

/*Add 3 tabs (the tabs are page (lv_page) and can be scrolled*/
lv_obj_t *tab1 = lv_tabview_add_tab(tabview, "Tab 1");
lv_obj_t *tab2 = lv_tabview_add_tab(tabview, "Tab 2");
lv_obj_t *tab3 = lv_tabview_add_tab(tabview, "Tab 3");


/*Add content to the tabs*/
lv_obj_t * label = lv_label_create(tab1, NULL);
lv_label_set_text(label, "This the first tab\n\n"
                         "If the content\n"
                         "become too long\n"
                         "the tab become\n"
                         "scrollable\n\n");

label = lv_label_create(tab2, NULL);
lv_label_set_text(label, "Second tab");

label = lv_label_create(tab3, NULL);
lv_label_set_text(label, "Third tab");
```

Sütun (lv_bar)

Written for v5.1

Genel Bakış

Bar nesnesi **arkaplan** ve **işaretleyici(gösterge)** olmak üzere iki parçadan oluşur. Nesnenin temeli **arkaplan**dır, **işaretleyici(gösterge)** arkaplan ile şekil olarak benzerdir fakat genişlik ve yüksekliği ayarlanabilir.

Bar'ın yönü genişlik/yükseklik oranına bağlı olarak **dikey veya yatay** olabilir. Mantıksal olarak yatay barlar genişlik, dikey barlar yükseklik değerleri değiştirilerek oluşturulur.

`lv_bar_set_value(bar, new_value)` fonksiyonu ile **yeni değer** atanır. Bu değer `lv_bar_set_range(bar, min, max)` fonksiyonu ile belirlenmiş **aralık** (minimum ve maximum değerler) arasında bir değerdir. Varsayılan aralık değerleri 1..100 arasındadır.

Güncel değerden istenilen değere geçiş **animasyon** lu biçinde olur. Bu durumda `lv_bar_set_value_anim(bar, yeni_deger, animasyon_zamani)` fonksiyonu kullanılır.

Biçim Kullanımı

- Nesnenin temeli **arkaplan** bu nedenle biçim elementleri kullanılır. Varsayılan biçin için `LV_STYLE_PRETTY` dir.

- **işaretleyici(gösterge)** arkaplanın benzeridir. Işaretleyici(gösterge) biçimleri `lv_bar_set_style_indic(bar,&style_indic)` fonksiyonu ile ayarlanır ve arkaplan ile arasında mesafe bırakmak için *hpad* and *vpad* biçim elementleri kullanılır. Varsayılan biçim olarak `LV_STYLE_PRETTY_COLOR` kullanılır.

Notlar

- İşaretleyici gerçek bir nesne değildir; sadece bar tarafından çizilmiştir.

Örnek
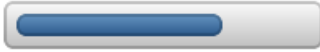
```c
/*Varsayılan Bar oluştur*/
lv_obj_t * bar1 = lv_bar_create(lv_scr_act(), NULL);
lv_obj_set_size(bar1, 200, 30);
lv_obj_align(bar1, NULL, LV_ALIGN_IN_TOP_RIGHT, -20, 30);
lv_bar_set_value(bar1, 70);

/*Bar'ın sağında etiket oluştur*/
lv_obj_t * bar1_label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(bar1_label, "Default");
lv_obj_align(bar1_label, bar1, LV_ALIGN_OUT_LEFT_MID, -10, 0);

/*Bar ve işaretleyici(gösterge) için biçim oluştur*/
static lv_style_t style_bar;
static lv_style_t style_indic;

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

lv_style_copy(&style_indic, &lv_style_pretty);
style_indic.body.grad_color =  LV_COLOR_GREEN;
style_indic.body.main_color=  LV_COLOR_LIME;
style_indic.body.radius = LV_RADIUS_CIRCLE;
style_indic.body.shadow.width = 10;
style_indic.body.shadow.color = LV_COLOR_LIME;
style_indic.body.padding.hor = 3;              /*İşaretleyici(gösterge)'yi biraz küçüklt*/
style_indic.body.padding.ver = 3;

/*İkinci bar oluştur*/
lv_obj_t * bar2 = lv_bar_create(lv_scr_act(), bar1);
lv_bar_set_style(bar2, LV_BAR_STYLE_BG, &style_bar);
lv_bar_set_style(bar2, LV_BAR_STYLE_INDIC, &style_indic);
lv_obj_align(bar2, bar1, LV_ALIGN_OUT_BOTTOM_MID, 0, 30); /*'Bar1''in altında hizala*/

/*Bar'ın sağında ikinci bir etiket oluştur*/
lv_obj_t * bar2_label = lv_label_create(lv_scr_act(), bar1_label);
lv_label_set_text(bar2_label, "Modified");
lv_obj_align(bar2_label, bar2, LV_ALIGN_OUT_LEFT_MID, -10, 0);
```

Takvim (lv_calendar)

Written for v5.2 (rev.4)

Genel Bakış

The Calendar object is a classic calendar which can: Takvim nesnesi klasik takvimdir:

- Şuanki gün ve haftayı vurgular

- Herhangi kullanıcı tanımlı tarihleri vurgular

- Günün ismini gösterir

- Buton tıklaması ile önceki/sonraki aylara geçiş sağlar

Takvimde tarihleri almak ve ayarlamak, `yıl` , `ay` and `gün` alanlarına sahip bir yapı olan `lv_calendar_date_t` tür kullanılır.

Mevcut tarihi ayarlamak için `lv_calendar_set_today_date(calendar, &today_date)` fonksiyonu kullanılır.

Gösterilen **tarihi ayarlamak için** `lv_calendar_set_shown_date(calendar, &shown_date)` kullanın;

Vurgulanan tarihlerin listesi bir `lv_calendar_date_t` dizisinde dizilmelidir ve bu diziyi geçerek `lv_calendar_set_hoghlighted_dates(calendar,` `&highlighted_dates)` 'e geçilebilir.

Dizinin yalnızca statik veya global bir değişken olması için yalnızca diziler işaretçisi kaydedilir.

Günlerin adları, `lv_calendar_set_day_names(calendar, gun_isismleri)` ile ayarlanabilinir; burada `gun_isismleri` , `const char *` `gun_isismleri[7] = {"Pazar", "Pazartesi", ...};`

Bir tarih seçmek için eylem `v5.3` 'te desteklenecek ve deneysel kullanım için şimdi `dev-5.3` dalında mevcut olacaktır.

Biçim Kullanımı

- **LV_CALENDAR_STYLE_BG** `body` özelliklerini kullanarak arka planın biçimi ve `metin` özelliklerini kullanarak veri numaralarının biçimi.

- **LV_CALENDAR_STYLE_HEADER** Geçerli yıl ve ayın görüntülendiği başlığın biçimi. `body` ve `metin` özellikleri kullanılır.

- **LV_CALENDAR_STYLE_HEADER_PR** Preslenmiş başlık biçimi, sonraki / önceki ay düğmesine basılıyor. `metin` özellikleri oklar tarafından kullanılır.

- **LV_CALENDAR_STYLE_DAY_NAMES** Gün isimlerinin biçimi. `metin` özellikleri gün metinleri tarafından kullanılır ve `body.padding.ver` gün isimlerinin üzerindeki alanı belirler.

- **LV_CALENDAR_STYLE_HIGHLIGHTED_DAYS** `text` özellikleri, öne çıkan günlerin biçimini ayarlamak için kullanılır

- **LV_CALENDAR_STYLE_INACTIVE_DAYS** `text` özellikleri önceki / sonraki ayın görünen günlerinin biçimini ayarlamak için kullanılır.

- **LV_CALENDAR_STYLE_WEEK_BOX** `body` özellikleri, hafta kutusunun biçimini ayarlamak için kullanılır

- **LV_CALENDAR_STYLE_TODAY_BOX** Bugün kutusunun biçimini ayarlamak için `body` ve `text` özellikleri kullanılır

Örnek

```c
/*Takvim nesnesi oluştur*/
lv_obj_t * calendar = lv_calendar_create(lv_scr_act(), NULL);
lv_obj_set_size(calendar, 240, 220);
lv_obj_align(calendar, NULL, LV_ALIGN_CENTER, 0, 0);

/*Geçerli hafta için biçim oluştur*/
static lv_style_t style_week_box;
lv_style_copy(&style_week_box, &lv_style_plain);
style_week_box.body.border.width = 1;
style_week_box.body.border.color = LV_COLOR_HEX3(0x333);
style_week_box.body.empty = 1;
style_week_box.body.radius = LV_RADIUS_CIRCLE;
style_week_box.body.padding.ver = 3;
style_week_box.body.padding.hor = 3;

/*Bugün için biçim oluştur*/
static lv_style_t style_today_box;
lv_style_copy(&style_today_box, &lv_style_plain);
style_today_box.body.border.width = 2;
style_today_box.body.border.color = LV_COLOR_NAVY;
style_today_box.body.empty = 1;
style_today_box.body.radius = LV_RADIUS_CIRCLE;
style_today_box.body.padding.ver = 3;
style_today_box.body.padding.hor = 3;
style_today_box.text.color= LV_COLOR_BLUE;

/*Vurgulanan günler için biçim oluştur*/
static lv_style_t style_highlighted_day;
lv_style_copy(&style_highlighted_day, &lv_style_plain);
style_highlighted_day.body.border.width = 2;
style_highlighted_day.body.border.color = LV_COLOR_NAVY;
style_highlighted_day.body.empty = 1;
style_highlighted_day.body.radius = LV_RADIUS_CIRCLE;
style_highlighted_day.body.padding.ver = 3;
style_highlighted_day.body.padding.hor = 3;
style_highlighted_day.text.color= LV_COLOR_BLUE;

/*Biçimleri uygula*/
lv_calendar_set_style(calendar, LV_CALENDAR_STYLE_WEEK_BOX, &style_week_box);
lv_calendar_set_style(calendar, LV_CALENDAR_STYLE_TODAY_BOX, &style_today_box);
lv_calendar_set_style(calendar, LV_CALENDAR_STYLE_HIGHLIGHTED_DAYS, &style_highlighted_day);


/*Bugün'ü ayarla*/
lv_calendar_date_t today;
today.year = 2018;
today.month = 10;
today.day = 23;

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

/*Vurgulanan bazı günler*/
static lv_calendar_date_t highlihted_days[3];        /*Sadece işaretçi kaydedilecek, bu yüzden statik olmalı*/
highlihted_days[0].year = 2018;
highlihted_days[0].month = 10;
highlihted_days[0].day = 6;

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

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

lv_calendar_set_highlighted_dates(calendar, highlihted_days, 3);
```

Tartı (lv_gauge)

Written for v5.1

Overview

The gauge is a meter with **scale labels** and **needles**. You can use the `lv_gauge_set_scale(gauge, angle, line_num, label_cnt)` function to adjust the scale angle and the number of the scale lines and labels. The default settings are: 220 degrees, 6 scale labels and 21 lines.

The gauge can show **more then one needles** . Use the `lv_gauge_set_needle_count(gauge, needle_num, color_array)` function to set the number of needles and an array with colors for each needle. (The array must be static or global variable).

You can use `lv_gauge_set_value(gauge, needle_id, value)` to **set the value of a needle**.

To set a **critical value** use `lv_gauge_set_critical_value(gauge, value)`. The scale color ill be changed to *line.color* after this value. (default: 80)

The **range** of the gauge can be specified by `lv_gauge_set_range(gauge, min, max)` .

Style usage

The gauge uses one style which can be set by `lv_gauge_set_style(gauge, &style)`. The gauge's properties are derived from the following style attributes:

- **body.main_color** line's color at the beginning of the scale

- **body.grad_color** line's color at the end of the scale (gradient with main color)

- **body.padding.hor** line length

- **body.padding.inner** label distance from the scale lines

- **line.width** line width

- **line.color** line's color after the critical value

- **text.font/color/letter_space** label attributes

Example
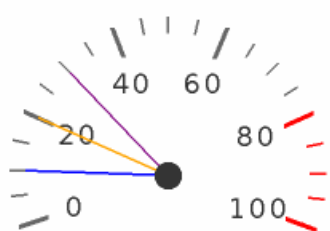
```
/*Create a style*/
static lv_style_t style;
lv_style_copy(&style, &lv_style_pretty_color);
style.body.main_color = LV_COLOR_HEX3(0x666);        /*Line color at the beginning*/
style.body.grad_color =  LV_COLOR_HEX3(0x666);       /*Line color at the end*/
style.body.padding.hor = 10;                         /*Scale line length*/
style.body.padding.inner = 8 ;                       /*Scale label padding*/
style.body.border.color = LV_COLOR_HEX3(0x333);      /*Needle middle circle color*/
style.line.width = 3;
style.text.color = LV_COLOR_HEX3(0x333);
style.line.color = LV_COLOR_RED;                     /*Line color after the critical value*/


/*Describe the color for the needles*/
static lv_color_t needle_colors[] = {LV_COLOR_BLUE, LV_COLOR_ORANGE, LV_COLOR_PURPLE};

/*Create a gauge*/
lv_obj_t * gauge1 = lv_gauge_create(lv_scr_act(), NULL);
lv_gauge_set_style(gauge1, &style);
lv_gauge_set_needle_count(gauge1, 3, needle_colors);
lv_obj_align(gauge1, NULL, LV_ALIGN_CENTER, 0, 20);

/*Set the values*/
lv_gauge_set_value(gauge1, 0, 10);
lv_gauge_set_value(gauge1, 1, 20);
lv_gauge_set_value(gauge1, 2, 30);
```

```
/*Create a style*/
static lv_style_t style;
lv_style_copy(&style, &lv_style_pretty_color);
style.body.main_color = LV_COLOR_HEX3(0x666);        /*Line color at the beginning*/
style.body.grad_color =  LV_COLOR_HEX3(0x666);       /*Line color at the end*/
style.body.padding.hor = 10;                         /*Scale line length*/
style.body.padding.inner = 8 ;                       /*Scale label padding*/
style.body.border.color = LV_COLOR_HEX3(0x333);      /*Needle middle circle color*/
style.line.width = 3;
style.text.color = LV_COLOR_HEX3(0x333);
style.line.color = LV_COLOR_RED;                     /*Line color after the critical value*/
```

Temel nesne (lv_obj)

Written for v5.1(rev.4)

Genel Bakış

Temel Nesne nesnelerin **en temel özelliklerini** içerir.

- Koordinatlar

- Kapsayıcı(Parent) Nesne

- Alt(Children)

- Biçim

- Klikleme aktif, taşıma aktif gibi özellikler.

Kapsayıcı(parent)'a bağlı olarak **x ve y koordinatları** `lv_obj_set_x(nesne, yeni_x)` ve `lv_obj_set_y(nesne, yeni_y)` fonksiyonları ile veya `lv_obj_set_pos(nesne, yeni_x, yeni_y)` bir fonksiyonu ile ayarlanabilir.

Nesne boyutu `lv_obj_set_width(nesne, yeni_genislik)` ve `lv_obj_set_height(nesne, yeni_yukseklik)` fonksiyonları veya `lv_obj_set_size(nesne, yeni_genislik, yeni_yukseklik)` fonksiyonu ile modifiye edilebilir.

Nesneyi `lv_obj_align(nesne1, nesne2, LV_ALIGN_TYPE, x_kaydir, y_kaydir)` fonksiyonu ile başka bir nesneye göre **hizalama**. Son iki argüman (x ve y kaydir) değerleri hizalama yapıldıktan sonra kaydırma değerleridir. İkinci argüman(nesne2) ise ilk argüman(nesne1)'ı hizalamada referans olarak kullanılır ve bu argümana(nesne2)'ye göre hizalama yapar, eğer `NULL` olarak tanımlanmış ise hizalama ekrana göre veya ayarlanmış kapsayıcıya göre uygulanır. Üçüncü eleman ise hizalama türürüdür:
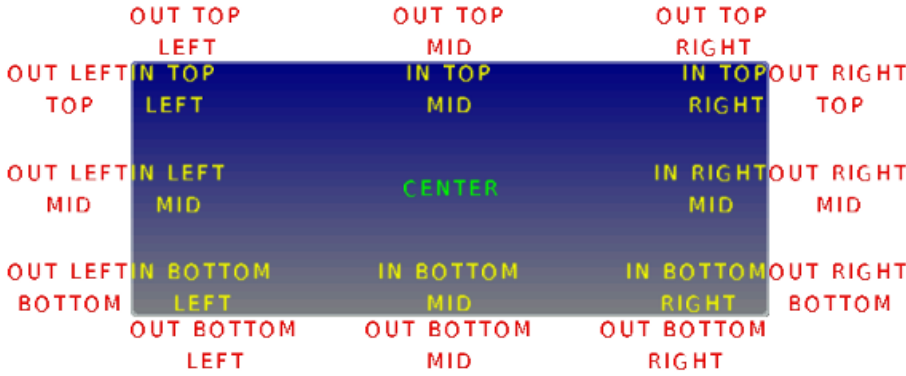
Hizalama türleri `LV_ALIGN_OUT_TOP_MID` bu şekilde oluşturulur. Örneğin bir yazıyı resim dosyasının altına 10 pixel sağa kaydırarak yazdırmak için `lv_obj_align(text, image, LV_ALIGN_OUT_BOTTOM_MID, 0, 10)` şeklinde kullanılır veya yazıyı(text)'ı kapsayıcının merkezinde yazdırmak için `lv_obj_align(text, NULL, LV_ALIGN_CENTER, 0, 0)` şeklinde kullanılır.

Nesne için `lv_obj_set_parent(nesne, yeni_kapsayıcı)` fonksiyonu ile **yeni kapsayıcı** ayarlayabilirsiniz.

Bir nesnenin altlarına sondan başa doğru gitmek için `lv_obj_get_child(nesne, alt_onceki)` fonksiyonu ile, baştan sona doğru gitmek için `lv_obj_get_child_back(obj, child_prev)` fonksiyonu kullanılır. İlk alt birimi geçip bir önceki dönüş değeri için ikinci parametre *NULL* seçilir. Fonksiyon *NULL* ile dönüyorsa başka bir alt birim yoktur.

`lv_obj_create(NULL, NULL)` komutu ile **ekran** oluşturduğumuzda, `lv_scr_load(ekran1)` fonksiyonu ile **yükleme** yapabiliriz. `lv_scr_act()` fonksiyonu **geçerli ekran**'na bir işaretçi tanımlar.

Otomatik olarak oluşturulan **iki katman** vardır:

- üst katman

- system katman

Ekranlardan bağımsız olarak oluşturulan nesneler herbir ekranda gösterilecektir. Üst katman(*top layer*) ekrandaki herbir nesnenin üzerindedir ve sistem katmanı(*system layer*) da üst katmanın üzerindedir. Üst katmana pop-up penceresi serbestçe eklenebilir fakat sistem katmanı sistem seviye işleri için sınırlandırılmıştır (örneğin fare imleci buraya taşınmıştır). `lv_layer_top()` ve `lv_layer_sys()` fonksiyonları üst veya sistem katman için bir işaretçi tanımlar.

Nesne için `lv_obj_set_style(obj, &yeni_biçim)` fonksiyonu ile **yeni biçim** tanımlanabilir, eğer biçim `NULL` seçildiyse nesne kapsayıcısının biçim mirasını alır. Eğer nesne için **modifiye bir biçim** kullanılacak ise kesinlikle bunu **nesneye bildirim** yapmamız gerekir.

`lv_obj_refresh_style(nesne)` fonksiyonu ile nesnenin biçimini yenileyebilir veya tüm nesnelere `lv_obj_report_style_mod(&biçim)` fonksiyonu ile bildirim yapılabilir. _lv_obj_report_style_mod_'s parametresini `NULL` olarak ayarlayıp tüm nesnelere bildirim yap.

`lv_obj_set_...(obj, true/false)` fonksiyonu ile Aktif/Pasif yapılabilen bazı özellikler:

- **hidden** Nesneyi gizle. Çizilmeyecek ve herhangi bir yer kaplamayacak. Alt elemanları da gizlenecek.

- **click** Nesnenin tıklama özelliğini bir girdi cihazı ile aktifleştir (örneğin dokunmatik). Eğer pasif ise, nesne arkasındaki giriş aygıtı tıklama işlemi sırasında kontrol edilir.

- **top** Eger aktifleştirilirse bu nesne veya nesnenin herhangi birine tıklandığında nesne ön tarafa doğru gelir.

- **drag** Sürüklemeyi aktifleştir (bir giriş aracı ile haraket ettirilir)

- **drag_throw** Sürükleme ile bırakmayı aftifleştirildiğinde nesne süskleme ile hızlanma kazanır.

- **drag_parent** Eğer aktifleştirilir ise nesnenin kapsayıcısıda sürüklenir.

Kütüphanede otomatik oluşan bazı özel aksiyonlar vardır, bu bir veya birden çok aksiyondan korunmak için **nesneyi koru** yapılmalı. Mevcut koruma seçenekleri:

- **LV_PROTECT_NONE** Koruma yok

- **LV_PROTECT_POS** Otomatik pozisyon koruma (örneğin. lv_cont)

- **LV_PROTECT_FOLLOW** Takip eden nesnelerin otomatik düzenleme koruma (örneğin. lv_cont)

- **LV_PROTECT_PARENT** Otomatik kapsayıcı değişim koruma

- **LV_PROTECT_CHILD_CHG** Alt değişim sinyal pasifleştirme. Kütüphane tarafından kullanılır.

`lv_obj_set/clr_protect(obj, LV_PROTECT_...)` fonksiyonları ile koruma ayarla/temizle yapılır. 'OR' kullanılarak koruma tip değerleri de kullanılabilir.

Nesneler için **gömülü animasyonlar** vardır. Mevcut animasyon türleri:

- **LV_ANIM_FLOAT_TOP** .. dan üst'e kaydır.

- **LV_ANIM_FLOAT_LEFT** .. dan sol'a kaydır.

- **LV_ANIM_FLOAT_BOTTOM** .. dan taban'a kaydır.

- **LV_ANIM_FLOAT_RIGHT** .. dan sağ'a kaydır.

- **LV_ANIM_GROW_H** Yatay Büyült/Küçült

- **LV_ANIM_GROW_V** Dikey Büyült/Küçült

`lv_obj_animate(obj, anim_type, time, delay, callback)` fonsiyonu nesneye animasyon uygular. Animasyon yönünü belirlemek için _ANIM_IN_ veya _ANIM_OUT_ animasyon türü seçilir, eğer tür belirtilmemişse varsayılan _ANIM_IN_ dir. Daha fazla bilgi için animations.

Biçim Kullanımı

Tüm *style.body* özellirleri kullanır. Normal bir nesne için varsayılan ekranlar _lv_style_plain_ and _lv_style_plain_color_.

Örnek

```
/*Basit temel nesne oluştur*/
lv_obj_t * obj1;
obj1 = lv_obj_create(lv_scr_act(), NULL);
lv_obj_set_size(obj1, 150, 40);
lv_obj_set_style(obj1, &lv_style_plain_color);
lv_obj_align(obj1, NULL, LV_ALIGN_IN_TOP_MID, 0, 40);

/*Bir önceki nesneyi kopyala ve sürüklemeyi aktifleştir.*/
lv_obj_t * obj2;
obj2 = lv_obj_create(lv_scr_act(), obj1);
lv_obj_set_style(obj2, &lv_style_pretty_color);
lv_obj_set_drag(obj2, true);
lv_obj_align(obj2, NULL, LV_ALIGN_CENTER, 0, 0);

static lv_style_t style_shadow;
lv_style_copy(&style_shadow, &lv_style_pretty);
style_shadow.body.shadow.width = 6;
style_shadow.body.radius = LV_RADIUS_CIRCLE;

/*Bir önceki nesneyi kopyala (sürükeleme önceden  aktif edilmiş)*/
lv_obj_t * obj3;
obj3 = lv_obj_create(lv_scr_act(), obj2);
lv_obj_set_style(obj3, &style_shadow);
lv_obj_align(obj3, NULL, LV_ALIGN_IN_BOTTOM_MID, 0, -40);
```

Yay (lv_arc)

Written for v5.2

Genel Bakış

Belirlenen **kalınlık** ile **başlangıç ve bitiş** açıları verilmiş bir **Yay nesnesi** çizilmesi.

Açıların ayarlanması için `lv_arc_set_angles(arc(yay), baslangic_acisi, bitis_acisi)` fonksiyonu kullanılır. Sıfır derece nesnenin altından başlar ve saat tersi yönünde devam eder, açı değerleri [0;360] arasındadır.

Yay nesnesinin biçim(style) ayarlaması için `lv_arc_set_style(arc, LV_ARC_STYLE_MAIN, &style)` fonksiyonu kullanılır.

Biçim kullanımı

- **line.rounded** bitiş noktasını yuvarlaştırmak için kullanılır (Şeffaflık(opacity) 1 yapılırsa bu özellik çalışmaz)

- **line.width** Yay kalınlığı.

- **line.color** Yay rengi.

Notlar

- Yay için **genişlik ve boy** aynı olmalı.

- Yay **anti-aliasing** i şuan **desteklememektedir**.

Örnek

```
/*Yay için biçim oluştur*/
lv_style_t style;
lv_style_copy(&style, &lv_style_plain);
style.line.color = LV_COLOR_BLUE;          /*Yay rengi*/
style.line.width = 8;                      /*Yay kalınlığı*/

/*Yay oluştur*/
lv_obj_t * arc = lv_arc_create(lv_scr_act(), NULL);
lv_arc_set_style(arc, LV_ARC_STYLE_MAIN, &style);          /*Yeni biçim kullanımı*/
lv_arc_set_angles(arc, 90, 60);
lv_obj_set_size(arc, 150, 150);
lv_obj_align(arc, NULL, LV_ALIGN_CENTER, 0, 0);

/*Bir önceki yay'ı kopyalayıp farklı açılarda ve boyutta yay oluştur*/
arc = lv_arc_create(lv_scr_act(), arc);
lv_arc_set_angles(arc, 90, 20);
lv_obj_set_size(arc, 125, 125);
lv_obj_align(arc, NULL, LV_ALIGN_CENTER, 0, 0);

/*Bir önceki yay'ı kopyalayıp farklı açılarda ve boyutta yay oluştur*/
arc = lv_arc_create(lv_scr_act(), arc);
lv_arc_set_angles(arc, 90, 310);
lv_obj_set_size(arc, 100, 100);
lv_obj_align(arc, NULL, LV_ALIGN_CENTER, 0, 0);
```

Yazı alanı (lv_ta)

Written for v5.1

Overview

The Text Area is **a page** with a **label** and a **cursor** on it. You can **insert text or characters** to the current cursor position with:

- `lv_ta_add_char(ta, 'c')`

- `lv_ta_add_text(ta, "insert this text")`

The `lv_ta_set_text(ta, "New text")` **changes the whole text**.

To **delete a character** from the left of the current cursor position use `lv_ta_del()`.

The cursor position can be modified directly like `lv_ta_set_cursor_pos(ta, 10)` or by stepping it:

- `lv_ta_cursor_right(ta)`

- `lv_ta_cursor_left(ta)`

- `lv_ta_cursor_up(ta)`

- `lv_ta_cursor_down(ta)`

There are several cursor types. You can set one of them with: `lv_ta_set_cursor_type(ta, LV_CURSOR_...)`

- LV_CURSOR_NONE

- LV_CURSOR_LINE

- LV_CURSOR_BLOCK

- LV_CURSOR_OUTLINE

- LV_CURSOR_UNDERLINE

You can 'OR' _LV_CURSOR_HIDDEN_ to any type to hide the cursor.

The Text area can be configures to be one lined with `lv_ta_set_one_line(ta, true)`.

The text area supports **password mode**. It can be enabled with `lv_ta_set_pwd_mode(ta, true)`.

Style usage

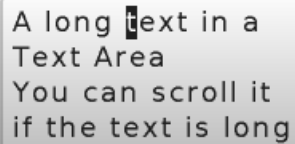Use `lv_ta_set_style(page, LV_TA_STYLE_..., &style)` to set a new style for an element of the text area:

- **LV_TA_STYLE_BG** background's style which uses all *style.body* properties. The label also uses this *style.label* from this style. (default: lv_style_pretty)

- **LV_TA_STYLE_SB** scrollbar's style which uses all *style.body* properties (default: lv_style_transp)

- **LV_TA_STYLE_CURSOR** cursor style. If NULL then the library sets us a style automatically according to the label color and font

  - LV_CURSOR_LINE: a *style.line.width* wide line but drawn as a rectangle as *style.body*. Hor. and ver. padding makes an offset on the cursor

  - LV_CURSOR_BLOCK: a rectangle as *style.body* Hor. and ver. padding makes the rectangle larger

  - LV_CURSOR_OUTLINE: an empty rectangle (just a border) as *style.body* Hor. and ver. padding makes the rectangle larger

  - LV_CURSOR_UNDERLINE: a *style.line.width* wide line but drawn as a rectangle as *style.body*. Hor. and ver. padding makes an offset on the cursor

Notes

- In password mode `lv_ta_get_text(ta)` gives the real text and not the asterisk characters

Example


```c
/*Create a scroll bar style*/
static lv_style_t style_sb;
lv_style_copy(&style_sb, &lv_style_plain);
style_sb.body.main_color = LV_COLOR_BLACK;
style_sb.body.grad_color = LV_COLOR_BLACK;
style_sb.body.border.color = LV_COLOR_WHITE;
style_sb.body.border.width = 1;
style_sb.body.border.opa = LV_OPA_70;
style_sb.body.radius = LV_RADIUS_CIRCLE;
style_sb.body.opa = LV_OPA_60;

/*Create a normal Text area*/
lv_obj_t * ta1 = lv_ta_create(lv_scr_act(), NULL);
lv_obj_set_size(ta1, 200, 100);
lv_obj_align(ta1, NULL, LV_ALIGN_CENTER, 0, - LV_DPI / 2);
lv_ta_set_style(ta1,LV_TA_STYLE_SB, &style_sb);              /*Apply the scroll bar style*/
lv_ta_set_cursor_type(ta1, LV_CURSOR_BLOCK);
lv_ta_set_text(ta1, "A text in a Text Area\n"
                "You can scroll it if the text is long enough.");   /*Set an initial text*/
lv_ta_set_cursor_pos(ta1, 2);                               /*Set the cursor position*/
lv_ta_add_text(ta1, "long ");                              /*Insert a word at the current cursor position*/

static lv_style_t style_bg;
lv_style_copy(&style_bg, &lv_style_pretty);
style_bg.body.shadow.width = 8;
style_bg.text.color = LV_COLOR_MAKE(0x30, 0x60, 0xd0);      /*Blue label*/

/*Create a one lined test are with password mode*/
lv_obj_t * ta2 = lv_ta_create(lv_scr_act(), ta1);
lv_obj_align(ta2, ta1, LV_ALIGN_OUT_BOTTOM_MID, 0, 50);
lv_ta_set_style(ta2,LV_TA_STYLE_BG, &style_bg);            /*Apply the background style*/
lv_ta_set_one_line(ta2, true);
lv_ta_set_cursor_type(ta2, LV_CURSOR_LINE);
lv_ta_set_pwd_mode(ta2, true);
lv_ta_set_text(ta2, "Password");
```