LVGL Framework User Guide

Version 1.0

Revision history

Version/Date	Modification
V1.0: 2023/10/10 Qianchao	- The first version
	-

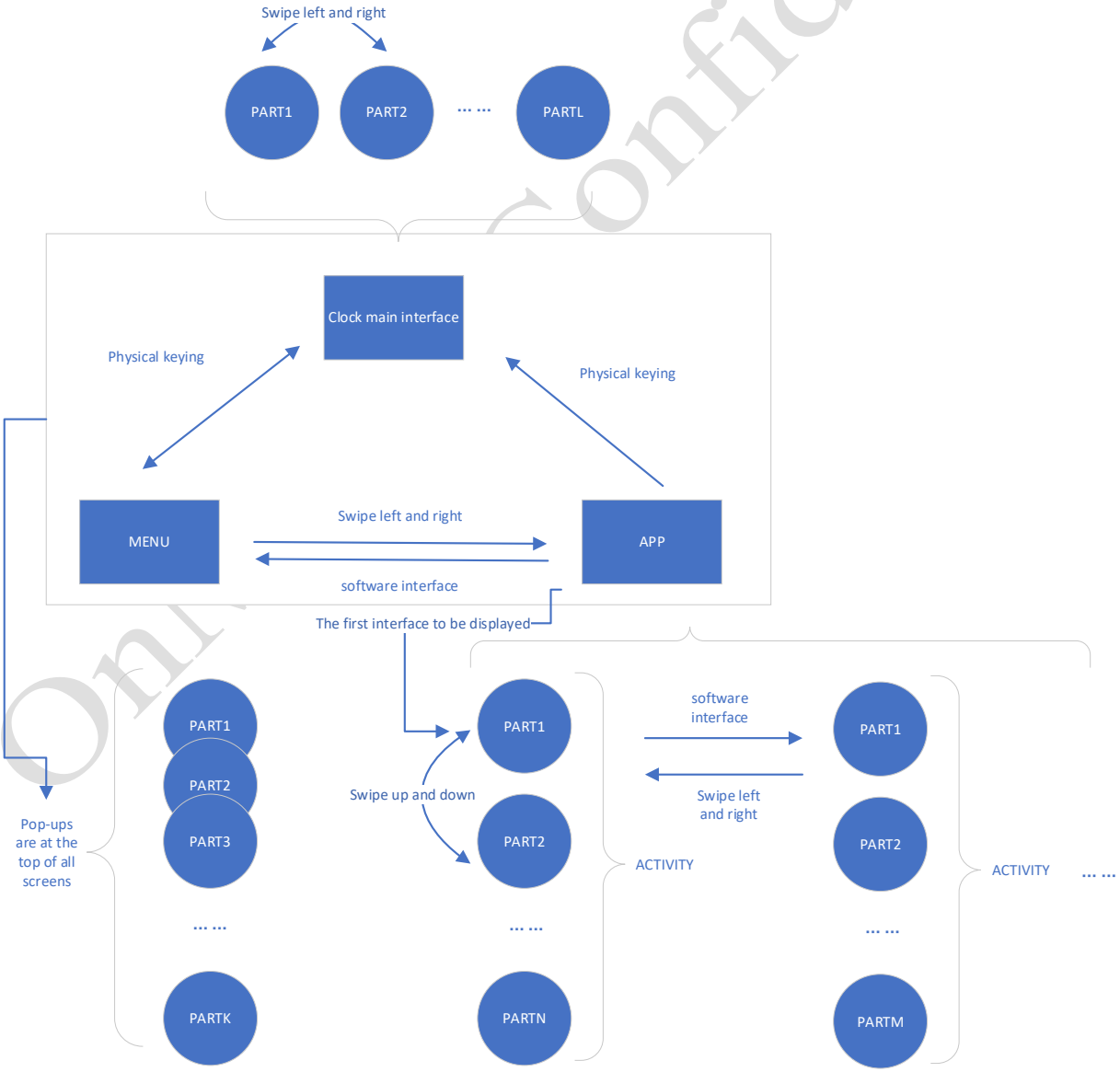
Contents

1 Introduction

1.1 Purpose and scope

It provides customers with a ready-made watch frame, which is convenient for customers to quickly make their own watch applications.
The framework is based on lvgl 8.2.0 library.
The framework interface function is divided into four parts, the main interface system, the menu interface, the app, and the pop-up window. Other features include hardware key interface, thread synchronization interface, monkey test and so on.

Correlation diagram of each screen interface



PART

Is the most basic component of any interface. There are create, start, stop, destroy and message synchronization functions.
1. create: Call lvgl to render the interface.
2. start: Starts the timer, monitors the message, etc.
3. stop: Stops the timer, monitors the message, etc.
4. destroy: Closes some messages, but instead of actively destroying LVGL-rendered ones, the framework 5.
5. automatically retrieves the entire interface content.
Synchronize messages: Prevents other threads from calling lvgl interfaces to lock causing deadlock problems.

APP

An APP consists of one or more activities（ACTS）, which can be switched through an interface. An ACT consists of one or more PARTs, which can be switched through up and down gestures. Jump to another ACT by calling the relevant interface.

MAIN interface

It is composed of multiple parts, which can be switched by sliding left and right, and can be set whether to scale and transparent during movement through the function interface.

MENU interface

Customers can sign up for the custom interface, which is the medium to access each APP.

Pop-up interface

Used when an "unexpected", immediate interface is needed, multiple can exist at once, with a priority, and "stacked" on other interfaces according to the priority. Like an incoming call.

thread synchronization

To prevent different threads from calling LVGL-related functions resulting in deadlock problems.

Monkey test

Three additional software simulation monkey tests are available:
1. Screen click: random coordinates, random time interval

2. Physical keys: Invoke the interface at random time
3. Gesture: random time interval, random starting point, random length, random direction

1.2 Acronyms and abbreviations

The table below defines the acronyms and abbreviations used in this document.

Acronym	Definition
LVGL	Light and Versatile Embedded Graphics Library
APP	application
ACT	activity

Table 1-1 Definitions and acronyms

1.3 References

2 Interface

2.1 APP

Enum and Define

om_app_return_dir_t

Returns the gesture of the previous ACT.

OM_APP_RETURN_DIR_RIGHT	Gesture to the right to return to the previous level
OM_APP_RETURN_DIR_LEFT	Gesture to the left to return to the previous level
OM_APP_RETURN_DIR_HOR	Compatible with the above two

Sturctures

om_app_version_t

APP version.

```
typedef struct {
    uint8_t major;
    uint8_t minor;
    uint8_t patch;
} om_app_version_t;
```

om_app_t

APP definition.

```
typedef struct _om_app_t{
    om_app_version_t version;       /**< The version of this APP */
    om_act_t        *act;           /**< The ACT that the APP displays first */
} om_app_t;
```

om_app_env_init_t

Environment variables required for APP framework initialization.

```
typedef struct {
  /// Set screen size
    uint16_t                glcd_x;
    uint16_t                glcd_y;
  /// Maximum number of layers of an APP
    uint16_t                acts_depth;
  /// Maximum number of the home screen
    uint16_t                cards_max;
  /// Whether to use additional features
    uint8_t                 extra_en;
} om_app_env_init_t;
```

Note: extra_en is a parameter that enables additional functionality. After opening, you need to initialize the structure om_app_extra_init_t. For details, see 2.2 struct om_app_extra_init_t.

Function Interface

om_app_init

Function：void om_app_init(om_app_env_init_t param);
Description： APP init.
Parameter：

param	initialize the required parameters

om_app_uninit

Function：void om_app_uninit(void);
Description： APP uninit.

om_app_create

Function：void om_app_create(om_app_t *app);
Description： create an APP.
Parameter：

app	describes an APP

om_app_destroy

Function：void om_app_destroy(void);
Description：destroy running APP.

om_app_set_return_dir

Function：void om_app_set_return_dir(om_app_return_dir_t dir);
Description：returns the direction of the gesture from the previous level.
Parameter：

dir	returns the gesture of the previous activity

om_app_register_menu_part

Function：void om_app_register_menu_part(om_part_t customer_part);
Description：register MENU interface.
Parameter：

customer_part	description of the menu interface

om_app_switch_position

Function：void om_app_switch_position(void);
Description：physical keys switch the location interface.

2.2 extra

Sturctures

om_app_extra_init_t

Frame extra function Settings.To use this function, enable it in om_app_init. And you need enough ram. See 3.1 for details `typedef struct {`

```
1.      /// Scale enable
2.     uint8_t     scale_en;
3.      /// Transparent enable
4.     uint8_t     trans_en;
5.      /// Transparency threshold
```

```
    uint8_t     trans_thresh;
} om_app_extra_init_t;
```

Function Interface

om_app_extra_init

Function：void om_app_extra_init(om_app_extra_init_t param);
Description：framework for additional functionality initialization.
Parameter：

param	functional parameter

om_app_extra_pr

Function：void om_app_extra_pr(lv_coord_t last_x, lv_coord_t last_y);
Description：gets the coordinate value at the time of rel.
Parameter：

last_x	x-coordinate
last_y	y-coordinate

om_app_extra_rel

Function：void om_app_extra_rel(lv_coord_t last_x, lv_coord_t last_y);
Description：gets the coordinate value at the time of rel.
Parameter：

last_x	x-coordinate
last_y	y-coordinate

om_app_extra_get_lv_timer_flag

Function：bool om_app_extra_get_lv_timer_flag(void);
Description：get timer status.
Return：

true	set
false	no-set

om_app_extra_get_read_timer_flag

Function：bool om_app_extra_get_read_timer_flag(void);

Description: gets the coordinate input timer status.
Return:

true	set
false	no-set

void om_app_extra_set_read_timer_flag

Function: void om_app_extra_set_read_timer_flag(bool flag);
Description: set the status of the coordinate input timer.
Parameter:

flag	set or not

om_app_extra_flush

Function: void om_app_extra_flush(void);
Description: fresh function.

2.3 act

Enum and Define

om_part_create_func_t

The PART creates a callback pointer.

```
1.  typedef void(*om_part_create_func_t)(void* scr);
```

om_part_destroy_func_t

The interface destroys the callback pointer.

```
2.  typedef void(*om_part_destroy_func_t)(void);
```

om_part_start_func_t

The interface starts the callback pointer.

```
3.  typedef void(*om_part_start_func_t)(void* scr);
```

om_part_stop_func_t

The partstops the callback pointer.

```
typedef void(*om_part_stop_func_t)(void);
```

om_part_handler_func_t

part processing message callback pointer.

```
typedef void(*om_part_handler_func_t)(uint16_t service_id, uint16_t evt_id, void *event);
```

Sturctures

om_part_t

Describes a PART.

```
typedef struct {
    om_part_create_func_t   create;  // necessary
    om_part_destroy_func_t  destroy;
    om_part_start_func_t    start;
    om_part_stop_func_t     stop;
    om_part_handler_func_t  handler;
    uint16_t                id;
} om_part_t;
```

om_act_t

Describes an ACT.

```
typedef struct {
     /// Number of PARTs
    int8_t                  parts_num;
     /// PARTs array
    const om_part_t       **parts;
} om_act_t;
```

Function Interface

om_act_switch

Function：void om_act_switch(om_act_t* om_act);
Description： switch to another ACT.
Parameter：

om_act	Describes an activity

om_act_set_mode

Function：void om_act_set_mode(om_act_mode_t mode);
Description： ACT set mode.
Parameter：

mode	whether the activity is bounded

om_act_switch_return

Function：void om_act_switch_return(void);
Description： manually return to the previous ACT.

om_act_restart

Function：void om_act_restart(void);
Description： restart the current ACT.

2.4 monkey

Function Interface

om_monkey_extra_physical_buttons

Function：void om_monkey_extra_physical_buttons(void);
Description： physical button monkey test starts

om_monkey_extra_gesture

Function：void om_monkey_extra_gesture(void);
Description： gesture monkey test initiated.

om_monkey_extra_get_point

Function：void om_monkey_extra_get_point(lv_coord_t* last_x, lv_coord_t* last_y);
Description： monkey test gets coordinates.
Parameter：

last_x	x-coordinate
last_y	y-coordinate

om_monkey_extra_get_state

Function：uint8_t om_monkey_extra_get_state(void);
Description： monkey test gain status.

Return：

state

2.5 main

Function Interface

om_main_create

Function：void om_main_create(void* scr, int32_t id);
Description： create the MAIN interface.
Parameter：

scr	pointer to scr
id	initialization position

om_main_set_parts

Function：void om_main_set_parts(om_part_t** parts, uint32_t num);
Description： set the card in the MAIN screen.

Parameter:

parts	part array
num	part num

2.6 meq

Enum and Define

om_app_popup_event_t

Returns the gesture of the previous ACT

OM_APP_POPUP_EVENT_FIRST	Pop-ups appear for the first time
OM_APP_POPUP_EVENT_ADD	Added a new pop-up window
OM_APP_POPUP_EVENT_DEL	Delete an existing pop-up
OM_APP_POPUP_EVENT_NOTHING	Delete the last pop-up

om_app_popup_cb_t

Popup callback function.

```
typedef void (*om_app_popup_cb_t)(om_app_popup_event_t event);
```

Function Interface

om_app_meq_handler

Function: void om_app_meq_handler(void);
Description: the lvgl framework processes all kinds of meq.

om_app_meq_malloc

Function: void* om_app_meq_malloc(size_t size);
Description: malloc the memory.
Parameter:

size	size

Return:

void*	pointer to memory

om_app_meq_free

Function：void om_app_meq_free(void *p);
Description： free the memory.
Parameter：

p	pointer to memory

om_app_meq_init

Function：void om_app_meq_init(void);
Description： message processing initialization

om_app_meq_add

Function：void om_app_meq_add(uint16_t app_id, uint16_t service_id, uint16_t evt_id, void *event);
Description： message added, waiting for processing.
Parameter：

app_id	app ID
service_id	service ID
evt_id	event ID
event	Pointer to event

om_app_handler_add

Function：void om_app_handler_add(uint16_t app_id, om_part_handler_func_t handler);
Description： add a processing callback.
Parameter：

app_id	app ID
handler	part handler function

om_app_popup_add

Function：void om_app_popup_add(uint16_t prio, uint16_t app_id, om_part_t* om_part);
Description： add popup.
Parameter：

prio	set the priority > 0
app_id	app ID
handler	describes an part

om_app_popup_del_app_id

Function: bool om_app_popup_del_app_id(uint16_t app_id);
Description: delete popup.
Parameter:

app_id	app ID

Return:

True	delete successfully
False	delete failed

om_app_popup_register_cb

Function: void om_app_popup_register_cb(om_app_popup_cb_t cb);
Description: register the popup event callback function.
Parameter:

cb	popup event callback

2.7 port

Function Interface

om_app_extra_flush_glcd_port

Function: void om_app_extra_flush_glcd_port(void* addr, lv_coord_t x1, lv_coord_t y1, lv_coord_t x2, lv_coord_t y2);
Description: glcd refreshes the interface.
Parameter:

addr	address
x1	the start coord of horizen
y1	the start coord of column
x2	the end coord of the horizen
y2	the end coord of the column

om_app_extra_scale_port

Function: void om_app_extra_scale_port(void* dst_addr, lv_coord_t dst_h, lv_coord_t dst_w, void* scr_addr, lv_coord_t scr_h, lv_coord_t scr_w);

Description: picture scaling interface.
Parameter:

dst_addr	destination address
dst_h	destination height
dst_w	destination width
scr_addr	original address
scr_h	original height
scr_w	original width

om_app_extra_blend_port

Function: void om_app_extra_blend_port(void* dst_addr, void* bg_addr, void* fg_addr, uint16_t fade_value);
Description: picture mixing interface.
Parameter:

dst_addr	destination address
bg_addr	bckground address
fg_addr	foreground address
fade_value	mixed number

om_app_extra_color_fill_port

Function: uint32_t om_app_extra_color_fill_port(void * dst, uint32_t value, uint32_t dest_stride, uint32_t w, uint32_t h);
Description: color filling.
Parameter:

dst	destination address
value	color value
dest_stride	filling stride
w	widget
h	hight

Return:

0	fail
other	address

om_app_extra_memcpy_stride_port

Function: void om_app_extra_memcpy_stride_port(void* dest_buf, const void* src_buf, uint16_t dest_stride, uint16_t src_stride, uint16_t w, uint16_t h);
Description: memcpy port.
Parameter:

dest_buf	destination address
src_buf	original address
dest_stride	destination stride
src_stride	original stride
w	widget
h	hight

om_app_extra_mem_malloc_port

Function: void* om_app_extra_mem_malloc_port(uint32_t size);
Description: request large memory SIZE > 5MB.
Parameter:

size	memory size

Return:

void*	pointer to memory

om_app_framework_log

Function: void om_app_framework_log(const char *fmt, ...);
Description: print log.
Parameter:

fmt	format
...	param

om_app_extra_init_port

Function: void om_app_extra_init_port(void);
Description: initializes extra functionality.

om_app_mem_malloc_general_port

Function: void* om_app_mem_malloc_general_port(uint32_t size);
Description: malloc the memory.
Parameter:

size	size

Return:

void*	pointer to memory

om_app_mem_free_general_port

Function：void om_app_mem_free_general_port(void* p);
Description： APP init.
Parameter：

p	pointer to memory

3 Example

3.1 Framework environment initialization

```
om_app_env_init_t param2 = {454, 454, 5, 5, 1};
om_app_init(param2);
```

Initialize the environment, for example, set the screen to 454 x 454, the maximum number of layers per APP to 5, the maximum number of PARTs on the home screen to 5, and enable additional functions.

3.2 Initializing Additional Functions

```
om_app_extra_init_t param1 = {1, 0, 100};
om_app_extra_init(param1);
```

Open additional functions, need to have enough ram space,

need >= (Screen length * Screen width * (bytes/pixels) *10)(byte),

open additional functions will make the sliding refresh speed is greatly improved, in addition, you can choose whether to turn on the zoom and transparency function.

Note: The function call after om_app_init().

3.3 Define an simple PART

The following example defines an simple PART that automatically reclaims control memory when leaving the PART frame.

```
#include "om_app.h"

static void xx_create(void *window)
{
    lv_obj_t *win = (lv_obj_t*)window;
    /// Set the background color black
    lv_obj_set_style_bg_color(win, lv_color_black(), 0);
    /// Create a label
    lv_obj_t *label = lv_label_create(win);
    lv_obj_set_style_text_color(label, lv_palette_main(LV_PALETTE_GREY), 0);
    lv_center(label);
    lv_label_set_text(reset_label, "hello");
}

static void xx_start(void *window)
```

```
{
    // Detect messages, use timers, etc
}

static void xx_stop(void)
{
    // Undetect messages, use timers, etc
}

const om_part_t app_xx_part = {
    .create = xx_create,
    .start = xx_start,
    .stop = xx_stop,
};
```

Note: The create callback is only relevant for rendering

3.4 Customizing the menu Interface

The user defines the icon form in the MENU and invokes the callback to launch the APP

```
extern om_app_t app_setting;
extern om_app_t app_activity;
extern om_app_t app_music;
extern om_app_t app_sleep;

static void btn_cb(lv_event_t *e)
{
    /// Create app in callback function
    om_app_t *om_app = lv_event_get_user_data(e);
    om_app_create(om_app);
}
 /// Set click callback
lv_obj_add_event_cb(btn0, btn_cb, LV_EVENT_CLICKED, &app_setting);
lv_obj_add_event_cb(btn1, btn_cb, LV_EVENT_CLICKED, &app_activity);
lv_obj_add_event_cb(btn2, btn_cb, LV_EVENT_CLICKED, &app_music);
lv_obj_add_event_cb(btn3, btn_cb, LV_EVENT_CLICKED, &app_heartrate);
```

The renderings are as follows:

3.5 MAIN Screen The initialization and Setting MENU screen is displayed

```
1.  /// What are the MAIN interfaces
2. om_part_t* app_main_parts[] = {&clock_screen1_part, &app_music_act_scr_part, &app_sleep_act
   _scr_part};
3. om_main_set_parts(app_main_parts, sizeof(app_main_parts) / sizeof(app_main_parts[0]));
4.  /// Initialization needs to be called once (only once) to display the MAIN screen
5. om_main_create(lv_scr_act(), 0);
6.  /// Set the MENU, the MENU is made by the customer himself
7. om_app_register_menu_part(app_main_menu_scr_part);
```

3.6 Interface Event Processing

When processing messages, the PART needs to call om_app_meq_add, add it to the message queue, and then process it in handler.

```
1. om_app_meq_add(APP_POPUP_ID_CALL_SCR_INCOMING, service_id, evt_id, event);
```

Note: It is used when other threads use the lvgl interface.

An PART with processing functions:

```
1. om_part_t app_telephone_incoming_scr_part = {
2.     .id         = APP_POPUP_ID_CALL_SCR_INCOMING,
3.     .create     = app_telephone_incoming_scr_create,
4.     .start      = app_telephone_incoming_scr_start,
5.     .stop       = app_telephone_incoming_scr_stop,
6.     .destroy    = app_telephone_incoming_scr_destroy,
7.     .handler    = app_telephone_incoming_scr_handler,
8. };
```

Note: If you have an id, you must have a handler. Each id is unique.

as follow:

```
enum {
    APP_PART_ID_RESERVED = 0,
    APP_FUNC_ID_BTN,
    APP_PART_ID_TEST,
    APP_PART_ID_CLOCK1_SCREEN,
    APP_PART_ID_MUSIC_ACT_SCREEN,
    APP_PART_ID_MUSIC_VOICE_SCREEN,
    APP_PART_ID_MUSIC_INQUIRY_SCREEN,
    APP_PART_ID_MUSIC_TYPE_SCREEN,
    APP_POPUP_ID_CLOCK,
    APP_POPUP_ID_CALL,
    APP_POPUP_ID_CALL_SCR_INCOMING,
    APP_POPUP_ID_CALL_SCR_OUTGOING,
    APP_POPUP_ID_CALL_SCR_ONGOING,
};
```

3.7 Composition of each APP

The following example shows that a sports APP consists of an ACT, which in turn consists of two PARTS. PART switches by sliding up and down, and swipes left and right to return to the MENU interface.

```
extern om_part_t app_activity_act_scr0_part;
extern om_part_t app_activity_act_scr1_part;
const om_part_t *app_act_scr_parts[] = { &app_activity_act_scr0_part ,&app_activity_act_scr1_part };

/// Describe an ACT
om_act_t app_activity_act = {
    .parts_num = 2,
    .parts = app_act_scr_parts,
};
 /// Describe an APP
const om_app_t app_activity = {
    .act = &app_activity_act,
};
```

The following example shows that the music APP consists of four ACTS, each of which is composed of one PART. Access to other ACTS is available through om_act_switch.

```
extern om_part_t app_music_voice_scr_part;
const om_part_t *app_music_voice_parts[] = { &app_music_voice_scr_part };
/// Describe an ACT
const om_act_t app_music_voice = {
    .parts_num = 1,
```

```
    .parts = app_music_voice_parts,
};

extern om_part_t app_music_type_scr_part;
const om_part_t *app_music_type_parts[] = { &app_music_type_scr_part };
/// Describe an ACT
const om_act_t app_music_type = {
    .parts_num = 1,
    .parts = app_music_type_parts,
};

extern om_part_t app_music_inquiry_scr_part;
const om_part_t *app_music_inquiry_parts[] = { &app_music_inquiry_scr_part };
/// Describe an ACT
const om_act_t app_music_inquiry = {
    .parts_num = 1,
    .parts = app_music_inquiry_parts,
};

extern om_part_t app_music_act_scr_part;
const om_part_t *app_music_act_parts[] = { &app_music_act_scr_part };
/// Describe an ACT
om_act_t app_music_act = {
    .parts_num = 1,
    .parts = app_music_act_parts,
};
 /// Describe an APP
const om_app_t app_music = {
    .act = &app_music_act,
};
```

Set the callback to go from the initial ACT app_music_act to app_music_voice

```
/// Set click event
lv_obj_add_event_cb(btn_volume, btn_volume_handler, LV_EVENT_CLICKED, NULL);
/// Toggle act in the callback
static void btn_volume_handler(lv_event_t *e)
{
    om_act_switch(&app_music_voice);
}
```

3.8 ACT Jump in APP

1. Go to the next level ACT

```
extern om_act_t app_music_voice;
static void btn_volume_handler(lv_event_t *e)
{
    om_act_switch(&app_music_voice);
}

 /// Called in part, clicking will take you to the next layer
lv_obj_add_event_cb(btn_volume, btn_volume_handler, LV_EVENT_CLICKED, NULL);
```

2. To return to the previous layer, you can call the interface void om_act_switch_return(void); Or swipe left and right.
3. To get to the same part of the ACT, swipe up and down.

3.9 Physical Keys Invoke the interface

You can use the synchronization mechanism provided by the framework. Exercise caution when using the mutex to prevent deadlock.

```
void om_app_switch_position(void);
```

3.10 Information Processing Registration

For some unexpected messages, such as phone calls, you need to pre-add handlers.

```
om_app_handler_add(APP_POPUP_ID_CALL, app_telephone_popup_handler);
```

3.11 Pop-up Use

According to the priority, create a pop-up window, the highest level will be displayed at the top, non-highest level will be sorted by priority and stop.

```
om_app_popup_add(prio, xx_id, &part);
```

The priorities are defined as follows:

```
enum {
    OM_APP_PRIO_APP         = 0, // minimum
    OM_APP_PRIO_POPUP_CLOCK,
    OM_APP_PRIO_POPUP_CALL,
};
```

Deletes the specified popup.

```
om_app_popup_del_app_id(xx_id);
```

3.12 Monkey test

Usage: Set the LV_USE_MONKEY macro in file lv_conf.h to 1.